A Novel Graphical Processing Unit Method for Power Systems Security Analysis

A DISSERTATION

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Laurie Elizabeth Miller

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

S. Massoud Amin, Bruce F. Wollenberg

June 2013

© Copyright Laurie Elizabeth Miller, June 2013

All Rights Reserved

Acknowledgments

I would like to thank Professors Massoud Amin and Bruce Wollenberg of the University of Minnesota Department of Electrical and Computer Engineering, for their knowledge, wisdom, guidance, and support.

Support for this work was provided by:

Principal Investigator	Duration	Award
Massoud Amin	3 years	Professor Amin's startup fund University of Minnesota
Massoud Amin	1.5 years	Oak Ridge National Laboratory (ORNL) Contracts 05226657 and 05226586
Massoud Amin	1 year	National Science Foundation Grant 0716253
Bruce Wollenberg	1 year	The University of Minnesota Center for Electric Energy

I would also like gratefully to acknowledge Dr. Arjun Shankar for his support and mentoring during my two summer appointments at ONRL. In addition I would like gratefully to acknowledge my introduction to scientific computing as an undergraduate honors and early masters student by Professor Matthew O'Keefe.

Abstract

There is an increasing need for computational power to drive software tools used in power systems planning and operations, since the emergence of modern energy markets and recent renewable generation technology fundamentally alters how energy flows through the existing power grid. While special-purpose hardware, including supercomputers, has been explored for this purpose, inexpensive commodity hardware is another way of getting increased computational power within the power systems control centers.

Adding General-Purpose Graphical Processing Units (GPGPUs) to the nodes in a control center's existing computational platform is a significantly lower expense than adding an equivalent number of new nodes and the infrastructure to support them. If accelerating computations with GPGPUs can halve the time needed for for a set of contingencies to run on a set of given computational nodes, freeing up crucial minutes for analysis of additional contingencies, the investment can be worth the costs. Yet this would be considered a quite modest speedup for GPGPU computing if the problem is conditioned in a way that maps well to the architecture and programming model of the GPGPU.

The novel method for GPGPU contingency analysis and its variants presented in this thesis allows that process of speedup to be taken substantially further, since it re-maps as much of the computation as possible to be a series of dense vector operations based on simple arithmetic that is conservative with respect to data movement and flexible with respect to implementation details such as thread block size. Where sparse matrix operations cannot be avoided, this method, by slicing across contingencies, re-maps such operations to

the much more efficient problem of a sparse matrix multiplied by a block of dense vectors larger than the matrix itself. The method applies to (N-1-1), (N-2), and (N-3) contingencies with little modification and little increase in computational burden or data movement per contingency. The method is designed to accommodate systems of thousands to tens of thousands of buses, if need be, with the large power systems resulting from control area consolidation in mind.

Contents

Acknowledgments **i**

Abstract **ii**

List of Tables **ix**

List of Figures **xi**

List of Acronyms and Abbreviations **xii**

1 Introduction: The Power Grid and Costs of Failures **1**

2 Power Systems Contingency Analysis **5**

2.1 Contingency Analysis and NERC Requirements 6

2.2 Uses of Power Systems Contingency Analysis 9

2.3 Large-Scale Contingency Analysis 13

2.4 Chapter Summary 19

3 General-Purpose Graphical Processing Units For Floating Point Acceleration **20**

3.1 Types of Parallel Computing 21

3.2 Parallel Computing with GPGPUs 23

3.3 A GPGPU as a Floating-Point Computation Accelerator 27

3.4 Current GPGPU Technology 32

3.4.1 Nvidia 32
3.4.2 AMD/ATI 36
3.4.3 Programming Models 36
3.4.4 Programming Standards 38
3.5 Chapter Summary 40

4 A Novel GPGPU Method for Power Systems Contingency Analysis 41

4.1 The Design Criteria for the Proposed Method 42
4.1.1 Characteristics and Limitations of GPGPU Technology 42
4.1.2 Design Criteria Chosen Due to Characteristics and Limitations of GPGPU Technology 47
4.1.3 Methods Chosen 50
4.2 The Fast Decoupled Power Flow 51
4.2.1 Power Flow Equations 51
4.2.2 Inputs to the FPDF 53
4.2.3 The $P - \theta$ Iteration: 55
4.2.4 The $Q - V$ Iteration: 56
4.3 Rectangular Form for the Current and Power Equations 58
4.3.1 Inputs to the FPDF 60
4.3.2 The $P - \theta$ Iteration: 61
4.3.3 The $Q - V$ Iteration: 62
4.4 Contingency Current updates 64
4.4.1 Basic derivation 64

4.4.2 Rectangular notation 66
4.4.3 Adding a shunt corrective term 67
4.4.4 Expressing the current updates as matrix operations 69
4.5 The Matrix Inversion Lemma 70
4.5.1 Definition 71
4.5.2 Sum of Two Matrices Case of the Matrix Inversion Lemma 71
4.5.3 Proof of the Sum of Two Matrices Case 72
4.5.4 Application of the Matrix Inversion Lemma to the FDPF Constant Matrices 73
4.5.5 Substituting the Matrix Inversion Lemma into the $\Delta\theta$ and ΔV Equations 75
4.5.6 Stott and Alsac Approximation 78
4.6 FPDF Contingency Algorithm with Stott and Alsac Approximation 86
4.6.1 Inputs to the FPDF Contingency Algorithm with Stott and Alsac Approximation 87
4.6.2 The $P - \theta$ Iteration: 88
4.6.3 The $Q - V$ Iteration: 90
4.7 FPDF Contingency Algorithm, Full $\Delta\theta$ and ΔV 92
4.7.1 Inputs to the FPDF Contingency Algorithm, Full $\Delta\theta$ and ΔV . . 93
4.7.2 The $P - \theta$ Iteration: 94
4.7.3 The $Q - V$ Iteration: 96
4.8 Summary 99

5 Details of the Proposed Method **100**

5.1 Common Routines 101

5.1.1 A Sparse Matrix Multiplied by a Block of Dense Vectors 101

5.1.2 Polar to Rectangular Conversion of a Block of Dense Vectors . . 102

5.1.3 Calculating the Bus Currents 109

5.2 Pre-Computation 118

5.2.1 The FDPF Base Case Constant Matrix Inverses 118

5.2.2 The $c_{B'}$ Constants and $x_{B'}$ Vectors 119

5.2.3 Computation of the $c_{B'}$ Constants and $x_{B'}$ Vectors 121

5.2.4 The $\Delta\theta$ Vectors 124

5.2.5 Pre-computation of the $\Delta\theta$ Vectors 126

5.2.6 The $c_{B''}$ Constants and $x_{B''}$ Vectors 128

5.2.7 Computation of the $c_{B''}$ Constants and $x_{B''}$ Vectors 130

5.2.8 The ΔV Vectors 133

5.2.9 Pre-computation of the ΔV Vectors 135

5.3 FPDF (N-1) Contingencies Algorithm 137

5.3.1 Inputs to the FPDF (N-1) Contingencies Algorithm 138

5.3.2 The $P-\theta$ Iteration: First Iteration 139

5.3.3 The $Q-V$ Iteration 167

5.3.4 The $P-\theta$ Iteration: Later Iterations 190

5.4 Computing Contingencies for (N-x) 210

5.4.1 Computing Contingencies for (N-1-1) 210

5.4.2 Computing Contingencies for (N-2) 214

5.4.3 Computing Contingencies for (N-3) 214

5.5 Summary 215

6 Recent Work in and Early Implementations of GPGPGU Contingency Analysis 216

6.1 Implementing Parts of a Power Flow on the GPGPU 217

6.2 Implementing a Power Flow on the GPGPU 218

6.3 Evaluating Power Flows in Parallel on the GPGPU 219

6.3.1 One Contingency Per Color Channel on a pre-CUDA GPU 219

6.3.2 One Power Flow Per GPGPU Thread Block 219

6.4 Summary 220

7 Conclusions 221

Bibliography 224

A Preliminary Results 235

B Compute Platform Details 238

B.1 Software 238

B.2 CPU Hardware 238

B.2.1 Output of NVIDIA CUDA's deviceQuery 239

B.2.2 Output of NVIDIA CUDA's bandwidthtest 242

B.2.3 Output of lspci 243

List of Tables

2 Elements used to form B' and B''[1]. 54

3 Formation of B' and B'' matrices [1]. 54

4 Differences in computational and data transport burden versus the set of (N-1) contingencies. 212

List of Figures

1 Basic reliability requirement from *NERC Policy 2- transmission* [2][3]. . . 7

2 Steps in static contingency analysis [4][5][6]. 10

3 Steps in dynamic contingency analysis [4][7]. 11

4 Number of contingencies for a given number of elements out of service. System sizes in number of elements, N, are plotted for N = 100, N = 1000, N = 8300, N = 17,000, N = 20,000, ... 15

5 Approximate number of minutes of computer time, without parallelism, to compute contingencies for a given number of elements out of service. System sizes in number of elements, N, are plotted for N = 100, N = 1000, N = 8300, N = 17,000, N = 20,000, ... 18

6 A general-purpose CPU block diagram. 22

7 Four classes of parallelism in computers, which together are known as Flynn's Taxonomy [8]. 23

8 A CPU augmented by a GPU, showing transistor area dedicated to each function [9]. 25

9 A layout of a typical GPGPGU [10]. 28

10 Growth of Nvidia GFLOPs rate scaled roughly according to Moores Law through 2011 [9]. 34

11 Nvidia has been aggressively increasing their theoretical maximum memory bandwidth [9]. 35

12 AMD/ATI GFLOPs rates have been scaling roughly according to Moores Law since 2005 [11]. 37

A.1 Timing results averaged over ten runs for vector itemwise multiplication (the vector inner product) for system sizes of 32, 200, 1024, 4096, 8192, and 16384 buses. 236

A.2 Timing results averaged over ten runs for multiplication of a sparse matrix by a set of dense vectors for system sizes of 512, 1024, 2048, 4096, and 8192 buses 237

List of Acronyms and Abbreviations

AC	Alternating Current
ALR	Adequate Level of Reliability
ALU	Arithmetic Logic Unit
AMD	Advanced Micro Devices
API	Applications Programming Interface
BPA	Bonneville Power Administration
CPU	Central Processing Unit
DC	Direct Current
DOE	Department of Energy
EIA	Energy Information Administration
EPRI	Electric Power Research Institute
ERCOT	Electric Reliability Council of Texas
FDPF	Fast Decoupled Power Flow
FERC	Federal Energy Regulatory Commission
FLOPS	Floating Point Operations Per Second
FPGA	Field-Programmable Gate-Array
GFLOPS	Giga-Floating Point Operations Per Second
GPU	Graphical Processing Unit
GPGPU	General-Purpose Graphical Processing Unit
IEEE	The Institute of Electrical and Electronics Engineers

MIMD	Multiple Instruction, Multiple Data-stream
MISD	Multiple Instruction, Single Data-stream
MISO	Midwest Independent System Operator
MRO	Midwest Reliability Organization
NERC	North American Electric Reliability Corporation
SCADA	Supervisory Control And Data Acquisition
SIMD	Single Instruction, Multiple Data-stream
SISD	Single Instruction, Single Data-stream
SMP	Streaming MultiProcessor
PES	Power and Energy Society (of the IEEE)
PF	Power Flow
VAR	Volt-Amperes Reactive (unit of reactive power)
WECC	Western Electricity Coordinating Council

Chapter 1

Introduction: The Power Grid and Costs of Failures

In darkened rooms across North America, highly trained specialists continually monitor the operating state of a machine the size of a continent. The computational power to provide these specialists all the information they would like to use has never been available, so they substitute insight, skill, and expert judgment to keep electric power available. Keeping power available is vital both economically and for the overall quality of life of those who use it. To accomplish that task, there is an increasing need for computational power to drive software tools used in power systems operations, since the emergence of modern energy markets and recent renewable generation technology fundamentally alters how energy flows through the existing power grid. While special-purpose hardware, including supercomputers, has been explored for this purpose, inexpensive commodity hardware is another way of getting increased computational power within the power systems control centers. Computational power and analysis tools are mechanisms that can help drive the larger goal to provide the social benefits of electrification to as many people as possible, as much of the time as possible.

Lack of electric power or poor quality of electric power costs in both money and quality of life. The National Academy of Engineering named the US power grid the greatest engineering achievement of the twentieth century [12]. However, blackouts and power quality issues in the United States are estimated to cost electricity customers $80 billion per year [13], and the true economic losses are estimated much higher, in the range of $119–188 billion per year [14][15][16]. The majority of these costs are not due to sustained blackouts, but rather to momentary interruptions or disruptions of power quality. For example, for some some manufacturing plants a momentary disruption may cause the same downtime and associated economic loss as a blackout several hours long [13]. Some sites do require uninterruptible power, and many industrial power customers would prefer to pay more for fewer power quality incidents [17]. The human toll is also high; for example, while air conditioning is often considered a luxury, for some it can be a dire necessity. More lives are lost in the United States in most years to hot weather than to all other types of natural disasters combined, and epidemiologists estimate that the number of heat-related deaths is in excess of 400 per year, possibly much higher [18]. The Chicago blackout and concurrent heatwave of 1995 killed over 500 in a single metropolitan area [19], with the deaths falling disproportionately on those in poverty. A blackout during a heatwave can be a disaster within a disaster. High temperatures place an even higher strain on the power grid suffering from increased air-conditioning load and heating of power system components, and these conditions are conducive to large cascading power failures. In areas not yet tied to a power grid, electricity could mean access to safe drinking water instead of the risk of dysentery [20]. An electric lamp that does not flicker means education becomes possible for adults and children who work as long as there is daylight [20]. In developed cities,

the working poor spend extremely long hours in work and transport, and quick access to information via the internet, from tax forms to bus schedules, becomes a necessity, not a luxury [21]. Hospitals can and do have their own backup generation within their facilities, but more and more people depend on home-based medical devices, such as Continuous Positive Airway Pressure (CPAP) machines, to improve quality of life [22]. But while new ways to improve quality of life using electric power are continually invented, the nature of the power grid itself is changing [23].

There is an increased and increasing need for computational power and speed for certain tasks in power systems operations. Power system operations centers use software tools to monitor the power system, and load flows and contingency analyses are part of the set of these tools [23][4][2][24]. These essential software tools are facing pressing computational needs, in part due to an expanding and more complex power system. For example, in the western interconnection of the North American power grid there are currently approximately 20,000 elements that must be accounted for in computations, and this number is growing [25]. Another change is that the incorporation of markets into power system operations has led to portions of the power grid operating very close to its limits at times for economic benefit [23]. While challenges such as the increasing size and complexity of the power grid would require increased computing speed for the software tools to run in the same time as previously, operating close to marginal limits requires that the tools be faster than before, since the shorter the time in which these tools take to run, the more closely and frequently the power grid can be monitored [23][2].

The increased need for computational power and speed can be met in a variety of

ways. One way of getting increased computational power within the power systems control centers is to use inexpensive commodity hardware that is cheaply and easily replaced. Supercomputers, possibly augmented with special-purpose processors, are another alternative [25][26]. Grid computing to augment existing computational platforms is yet another [23]. Where data security is crucial to protect market competitors from potentially gaining access to sensitive information, computing platforms that can be housed within the operations facility may be preferred. In that case, farming out computation to grid computing or off-site supercomputers would not be preferred. Solutions within the operations center include special-purpose hardware, such as supercomputers or special-purpose boards used to augment commodity hardware, or implementations with commodity hardware only. Special-purpose hardware is built in smaller quantities than commodity hardware, and is often more reliable, while commodity hardware is produced in mass quantities intended to be replaced relatively cheaply and easily [27][28]. Both have their advantages. Special-purpose hardware tends to deliver better performance when chosen correctly for the task at hand, while commodity hardware can be orders of magnitude cheaper and malfunctioning parts can be quickly changed out without calling in a special technician, which is what must often be done for special-purpose hardware.

While the impact of lack of high-quality electric power on economics and quality of life is the driving force for the work contained in this dissertation, the ensuing chapters will discuss in more detail the increased need for computational power and speed for certain tasks in power systems operations, and a novel approach for getting that computational power using inexpensive commodity hardware that is cheaply and easily replaced.

Chapter 2

Power Systems Contingency Analysis

Power systems contingency analysis forms one part of power systems security analysis along with power systems state estimation and other security analysis tools. Power systems security analysis involves tools, regulations, and practices to keep the power system operating when elements of the system fail [4][5]. The power system remains in a secure state if, for example, a generating unit is taken out of service but enough spinning reserve is maintained for all the load to still be served and for the system frequency to be maintained within its limits. Similarly, if a transmission line is taken out of service by a storm, the power system remains in a secure state if the power can still be delivered to the loads on the remaining lines with no violations of line loading parameters. In the United States and Canada, rules for maintaining the power grid in a secure state are set by the North American Electric Reliability Corporation (NERC). Both static and dynamic power systems security analyses are used to help meet those requirements over multiple timeframes from planning to real-time operation. The time constraints of power systems security analysis combined with limited available processing power limit the contingencies studied in real time operations. Since only a subset of the contingencies of greatest concern can be evaluated in dynamic security analysis for power systems operations, methods which allow for more contingencies to be evaluated in the time available are desired.

2.1 Contingency Analysis and NERC Requirements

Power system security is the ability of the electric power grid to withstand sudden disturbances and outages. A contingency is defined as a set of power system elements (including lines, transformers or generators) that are out of service but which would be in service in the normal system state. Power system security analysis studies and evaluates all contingencies, identifying those which may have significant consequences, such as leading to a cascading blackout [4][2][5]. Power system events may unfold faster than an operator can intervene to ensure system security [4][2]. If a transmission line goes out of service due to contact with vegetation, the power that was flowing on that line instantaneously flows on alternative paths instead, which can lead to line overloading, voltage violations, changes in system frequency, uncontrolled separation of parts of the power grid, and additional lines going out of service due to these issues in a cascading outage or cascading blackout [2][29][30]. Generators and loads may go out of service as well. Power system operators strive to manage their control areas such that any single contingency – the loss of any one element, such as a generator, transmission line, or transformer – will not result in a cascading blackout. This is known as the (N-1) criterion [4][2][29][7][31].

NERC sets rules for power system security and specifies requirements for handling power system contingencies. Contingencies of concern include not only single contingencies, in which one element is out of service, but multiple contingencies in which multiple elements are out of service at one time. The requirements set by NERC include setting the maximum allowable time frame after a contingency occurs after which the power system

must again be in a normal and secure operating state, able to withstand any single contingency. In the period immediately following a contingency the power system may or may not be able to withstand additional contingencies, and action must be taken to restore the power system to an operating state that can once again withstand any single contingency. Since the power system is in a state of increased vulnerability during the time from when a contingency occurs to when the system is again in a secure operating state, the duration of that time period must be kept to no longer than 30 minutes as shown in Figure 1, if not substantially less [2][3].

The current NERC basic reliability requirement from *NERC Policy 2- transmission*, effective July 1, 1998, is quoted here from [2][3], including Figure 1:

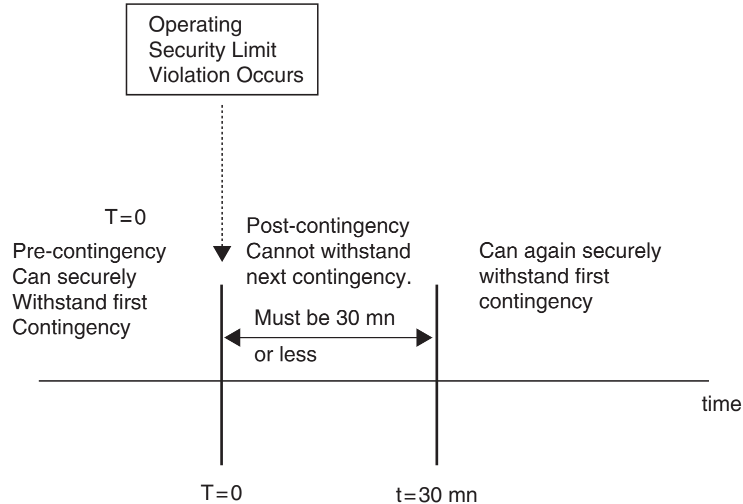

Figure 1: Basic reliability requirement from *NERC Policy 2- transmission* [2][3].

> 1. Basic reliability requirement regarding single contingencies: All control areas shall operate so that instability, uncontrolled separation, or cascading outages will not occur as a result of the most severe single

contingency.

1.1. Multiple contingencies: Multiple outages of credible nature, as specified by regional policy, shall also be examined and, when practical, the control areas shall operate to protect against instability, uncontrolled separation, or cascading outages resulting from these multiple outages.

1.2. Operating security limits: Define the acceptable operating boundaries.

2. Return from operating security limit violation: Following a contingency or other event that results in an operating security limit violation, the control area shall return its transmission system to within operating security limits soon as possible, but no longer than 30 minutes [2][3].

NERC further defines an Adequate Level of Reliability (ALR), stating that the bulk electric power system achieves an ALR when it possesses the following characteristics quoted here from [32]:

Characteristics of Adequate Level of Reliability (ALR):

1. Controlled to stay within acceptable limits during normal conditions
2. Perform acceptably after credible contingencies
3. Limit the impact and scope of instability and cascading outages when they occur
4. Facilities are protected from unacceptable damage by operating them within facility ratings

5. Integrity can be restored promptly if it is lost
6. Have the ability to supply the aggregate electric power and energy requirements of the electricity consumers at all times, taking into account scheduled and reasonably expected unscheduled outages of system components

The definition of ALR is being updated at the time of writing, with a draft available that includes socio-economic impact considerations [32].

2.2 Uses of Power Systems Contingency Analysis

Power systems security analysis methods and tools used to meet NERC requirements include both static and dynamic security analyses [4][2][7]. Static security analysis uses a snapshot of the system and its conditions taken when the system was in a stable state. This snapshot consists of a set of values for operating variables that defines the system state at the time of the snapshot and can be used for contingency analysis, iterating through each of a set of possible contingencies, and performing a power flow calculation and possibly other analyses as well [Figure 2]. The power system state information taken from the system snapshot along with a list of contingencies to be studied serve as the input data. A contingency is selected and the power system state information is updated to remove the appropriate element or elements from service. A power flow calculation is then carried out and checks are performed as to whether operating constraints on bus voltages or line flows are violated. If there are any violations, alarms are generated showing that the contingency has issues of concern. The procedure continues for each contingency in the list of those to

be studied [4][5][6].

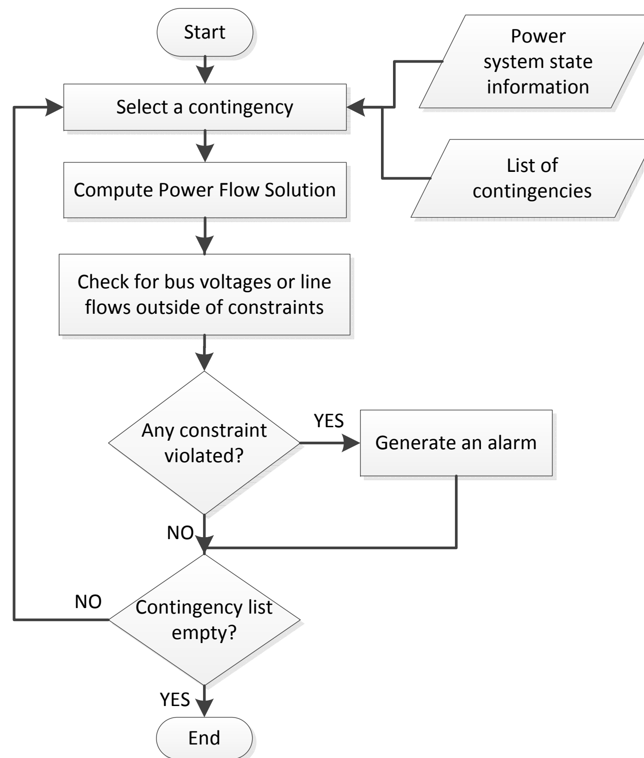

Figure 2: Steps in static contingency analysis [4][5][6].

Dynamic security analysis uses real-time field data that is acquired through the Supervisory Control And Data Acquisition (SCADA) system and fed into a software tool that performs power system state estimations to produce values for the same set of operating variables for the power system used in a static security analysis [4][2][7]. Figure 3 shows the steps in this process. Field data is acquired and fed into the power system state estimation computation. The resulting power system state information, along with a list of contingencies to be studied, serves as the input data to the contingency analysis, which then proceeds with the same steps used for static security analysis. A contingency is selected

and the power system state information is updated to remove the appropriate element or elements from service. A power flow calculation is then carried out and checks are performed on whether operating constraints on bus voltages or line flows have been violated. If there are any violations, alarms are generated showing that the contingency has issues of concern. The procedure onward from selecting the next contingency is repeated for each contingency in the list of those to be studied [4][7].

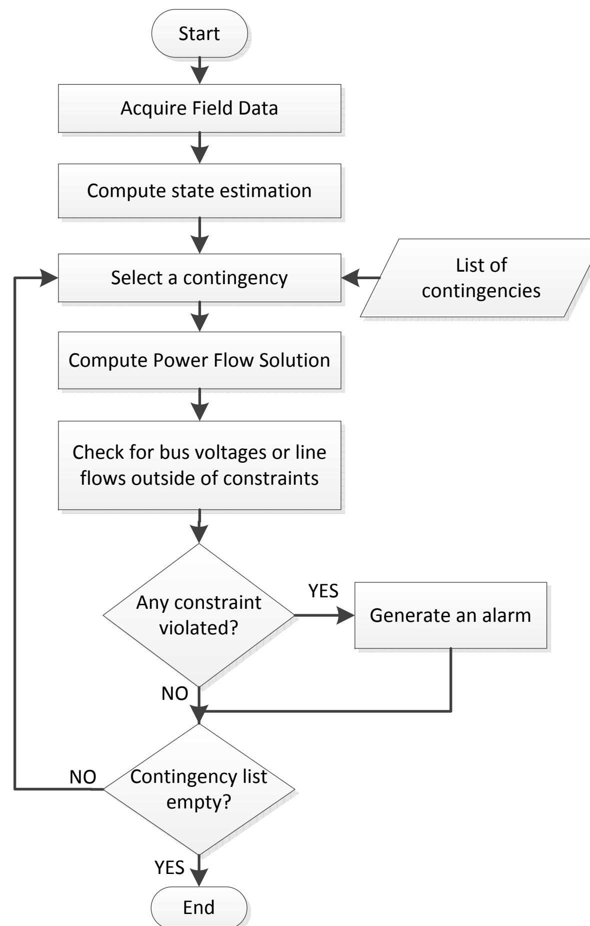

Figure 3: Steps in dynamic contingency analysis [4][7].

Power systems contingency analysis is used in three main time frame scenarios: 1) transmission and facilities planning, 2) power systems operational planning, and 3) power systems operations. In transmission and facilities planning, decisions are made for augmenting or changing the existing power grid to enhance the security and reliability of the system. Static security analysis focused on contingency analysis is used to test and select designs intended to be stable and secure in operation. However, years take place between the planning studies and the implementation of the planned changes to the power grid. The power grid does not remain static during this time, during which new loads are added, equipment is replaced or upgraded, and other changes are implemented. The second time frame is for power systems operational planning, which is done much more frequently to update the operating rules for a given control area. For operational planning, static security analysis focused on contingency analysis is used, but using a snapshot of the system state taken much closer in time to when the operational rules will be implemented, and this analysis forms part of the overall operations planning and studies. The third time frame is power systems operations, in which power systems operators deal with contingencies as they happen. Dynamic security analysis using state estimation to feed the contingency analysis is used to re-analyze the security of any new state of the power system as it arises [2][7].

In dynamic security analysis for power systems operations, the time window in which the security analysis must be completed is between 10 and 30 minutes [2], with results from state information only a few minutes old being more useful than results from data even minutes older [7]. There are three main approaches to performing the dynamic security analysis within the necessary time window [4]:

- Use fast approximations. This may include using a DC power flow instead of an AC power flow calculation [4][5][33], or other methods for arriving at a fast approximation such as calculating power flow on a reduced system model [5][34][35] [36][37], though these approximations may fail to capture contingencies of concern [4][5].

- Analyze a smaller number of contingencies by selecting those believed to be most critical [26] [6][33][34][35][36][37][38][39] [40][41][42][43][44][45][46][47][48]. Contingency screening or contingency selection methods help select the most necessary cases to analyze in the allotted time, but there still may be insufficient time to analyze the cases of interest [25][31].

- Make the computational platform faster. As discussed in Chapter 1, this can be done though using computing power located remotely or locally, using special-purpose hardware or mass-produced commodity hardware [23][26][25]. The focus of this work is on methods for using inexpensive commodity hardware.

2.3 Large-Scale Contingency Analysis

In the past ten years, many control areas in North America have been consolidated into larger balancing authorities, increasing the problem size for both static and dynamic security analysis. For example, the Midwest Independent System Operator (MISO) service territory was once divided into 26 control areas but is now operated as a single balancing authority. The MISO network model used for dynamic security analysis contains 40366 buses and 8300 contingencies in its pre-selected contingency list [49]. Dynamic security

analysis is usually conducted only on the elements within a particular control area, which does not account for failures in multiple control areas occurring simultaneously withing a single interconnect [25][31]. One possibility is to model an entire interconnect, such as the Western Interconnection of the North American power grid, which has approximately 17,000 elements [26] to 20,000 elements [25] to account for in contingency analysis. Both enlarging the list of contingencies analyzed and enlarging the area studied increases the amount of computation that is needed within the desired time window for dynamic security analysis. In addition to studying single contingencies, it may be desirable to study contingencies with greater numbers of elements taken out of service [25][7][31], which further enlarges the list of contingencies to be evaluated. Figure 4 gives an idea of the growth of the problem size and therefore the magnitude of computation required for systems of increasing size and for expanding the contingency list beyond single contingencies.

Figure 4 shows the number of contingencies for the (N-1) case, the (N-1-1) case, and the cases for (N-2), (N-3), (N-4), and (N-5). The set of (N-1) contingencies is the set of all contingencies such that one element (line, transformer, or generator) is removed from service. The number of (N-1) contingencies in a system with N elements is equal to N. The set of (N-1-1) contingencies is the set of all contingencies such that two elements are taken out of service consecutively. In other words, the set of (N-1-1) contingencies is the set of all (N-1) contingencies such that an additional element is removed from service. The number of (N-1-1) contingencies in a system with N elements is equal to N(N-1). The set of (N-2) contingencies is the set of all contingencies such that any two elements are removed from service simultaneously. The number of (N-2) contingencies in a system with N elements is equal to N(N-1)/2, half the number of (N-1-1) contingencies. For the set of

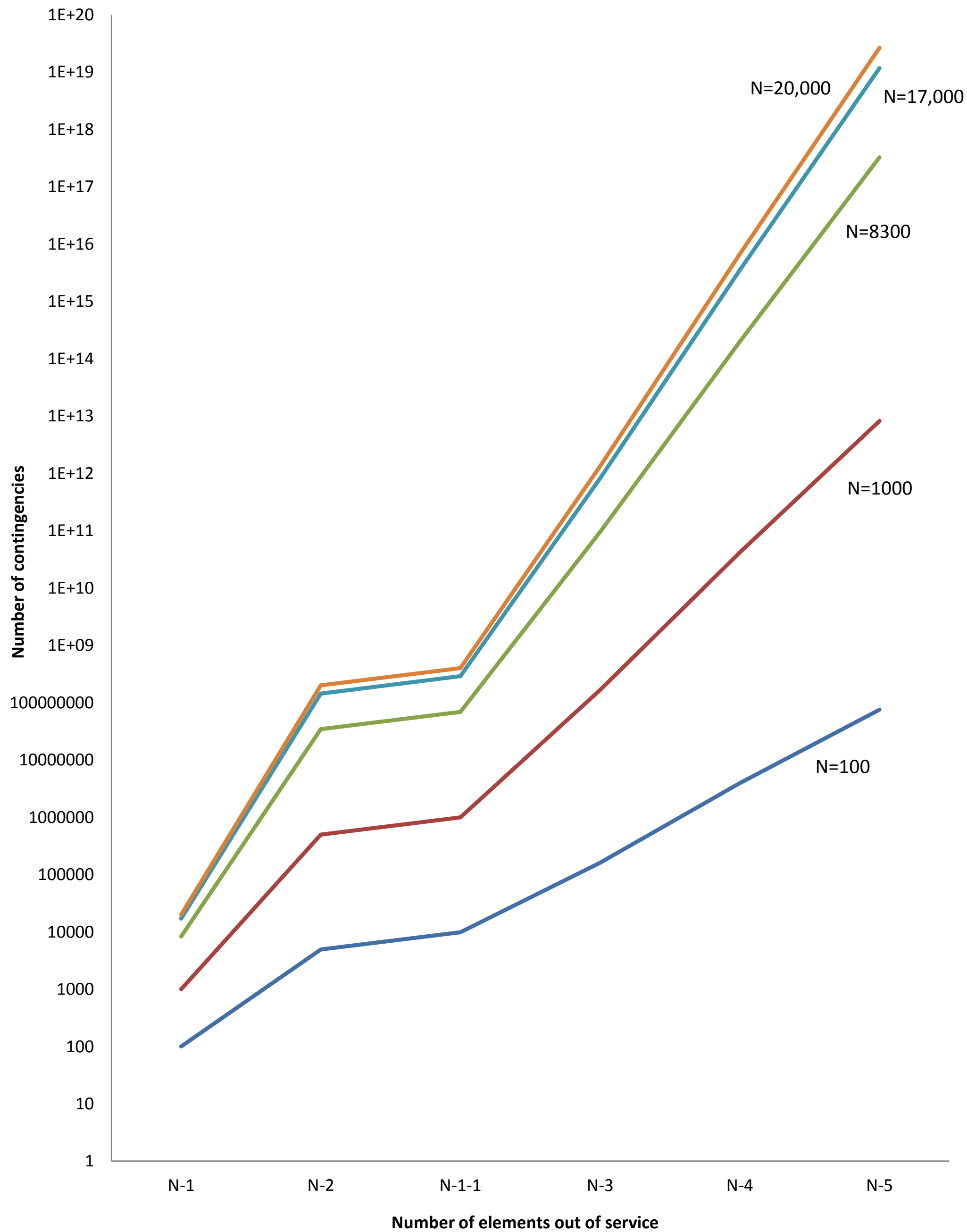

Figure 4: Number of contingencies for a given number of elements out of service. System sizes in number of elements, N, are plotted for N = 100, N = 1000, N = 8300, N = 17,000, N = 20,000,

(N-1-1) contingencies the order in which two elements are removed from service represent two distinct contingencies, whereas for the set of (N-2) contingencies, both elements are removed at the same time, and the order in which the removed elements are listed does not distinguish separate contingencies [4][50]. The set of (N-3), (N-4), and (N-5), contingencies is the set of all contingencies such that any three, four, or five elements respectively are removed from service simultaneously. Curves are given for N=20,000 and N=17,000, the approximate sizes of the Western Interconnection [25][26], N=8300, as an approximation of the number of elements in the MISO model (it is actually much larger) [49], and N=1000 and N=100 for illustrative purposes.

While available computing power has greatly increased in recent years, power systems control centers still commonly face having to run a reduced set of contingencies for dynamic security analysis in order for the computation to be completed in the required timeframe. Bonneville Power Administration (BPA) runs a pre-selected list of 500 contingencies every five minutes [29]. Hydro-Quebec still requires 10 to 30 minutes to run a full list of contingencies and therefore runs a reduced list of 1000 contingencies every three minutes [51]. MISO runs a reduced list of 8300 contingencies, but even the reduced list would require 20 minutes of serial computation, so parallel computation is employed to bring the execution time to 4.5 minutes [51]. The dynamic security analysis model for the Electric Reliability Council of Texas (ERCOT) contains 3938 contingencies, but to meet dynamic security analysis time constraints, a reduced list of 500 contingencies are selected via screening for full AC analysis [52].

Similar problems face private utilities as well. Nstar, founded over 100 years ago, serves 1.1 million electric customers in 81 communities and 300,000 natural gas customers

in 51 communities in eastern, central, and southeastern Massachusetts including the Boston urban area. In a 2010 merger, Nstar became an operating company of Northeast Utilities. The dynamic security analysis model at NStar would have about 3000 single contingencies, so a reduced list of 760 contingencies is selected to meet the time constraint of three minutes. While (N-1-1) contingencies are of interest to NStar, to deal with processing constraints they are run once an hour with the full contingency list [53].

Since elements from multiple control areas may fail simultaneously, it is desirable to perform security analysis on an entire interconnect. This is currently done for ERCOT, the smallest North American Interconnection [25][26]. For example, the Western Interconnection of the North American power grid contains 35 separate control areas, which makes it likely that there may be simultaneous contingencies occurring but in separate control areas. However, since the Western Interconnection, while smaller than the Eastern Interconnection, still contains on the order of 20,000 elements that must be modeled in contingency analysis, the computational time is considerable even at supercomputer speeds. Estimating 1/2 second of computing time per contingency means it would take 10,000 seconds, or just under three hours, to run the set of (N-1) contingencies for the Western Interconnection without parallel computing methods [25]. Figure 5 shows the growth of computing time required in minutes for the cases in Figure 4.

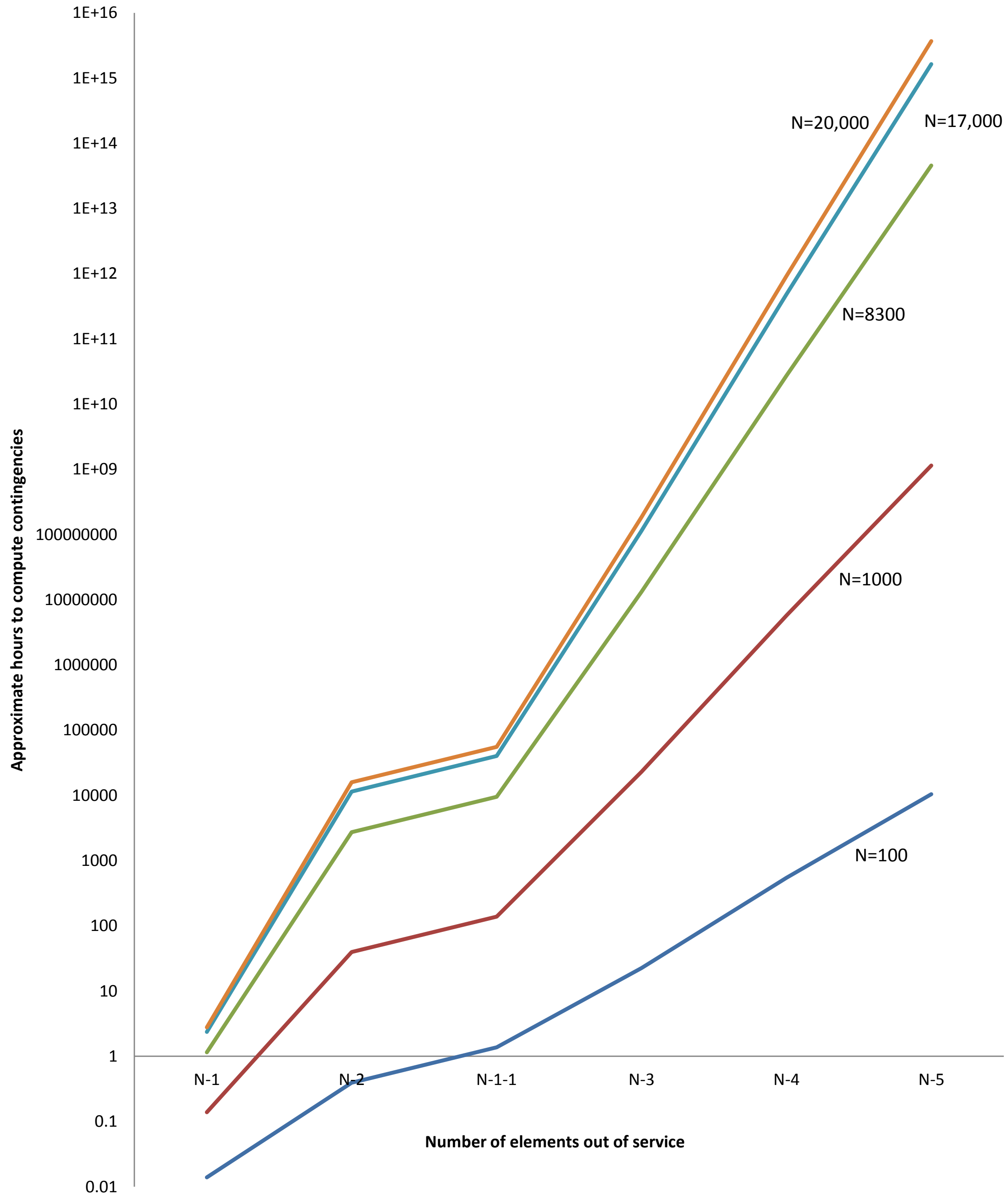

Figure 5: Approximate number of minutes of computer time, without parallelism, to compute contingencies for a given number of elements out of service. System sizes in number of elements, N, are plotted for N = 100, N = 1000, N = 8300, N = 17,000, N = 20,000,

2.4 Chapter Summary

Improved computational speed for power systems contingency analysis remains an ongoing problem in the North American power grid. The computational power to run exhaustive sets of contingencies for the entire Eastern and Western interconnects is not yet readily available, even for static security analysis used for long-term planning. To meet operational and NERC requirements for speed of dynamic security analysis, control centers run reduced sets of contingencies, while larger sets of contingencies may be run on longer timeframes for operation planning. Meeting time and computational constraints may also require using approximate calculations and using approximate, reduced system models, sacrificing accuracy for the necessary speed. Methods which allow larger models, more accurate computational methods, and very large numbers of contingencies in minutes are of pressing interest to power systems control centers.

Chapter 3

General-Purpose Graphical Processing Units For Floating Point Acceleration

Chapter 2 discussed the ongoing need to make power systems contingency analysis faster, and Section 2.2 lists three main approaches to accomplishing this; 1) use approximate calculations, 2) analyze a smaller number of contingencies, and 3) make the computational platform faster. For the third option, making the computational platform faster, Chapter 1 discussed options such as off-site supercomputers, grid computing, on-site special purpose hardware, and on-site commodity hardware. This work focuses on methods for using a specific type of on-site commodity hardware, Nvidia General-Purpose Graphical Processing Units (GPGPUs) to accelerate computation for power systems contingency analysis. GPGPUs both operate in parallel with a Central Processing Unit (CPU) and form a type of parallel processing unit in and of themselves, so this chapter begins with a discussion of the types of computational parallelism, then continues with the types of computational parallelism GPGPUs make possible.

3.1 Types of Parallel Computing

The part of a modern computer that actually performs the computing, the CPU, is a very small fraction of the size and weight of the computer system as a whole, and yet it contains complex structures within it. Further, the variety of manners in which CPUs can be combined for parallel computing results in a rich taxonomy of parallelism classifications.

A computer system CPU fetches program instructions and data from the computer system memory, then executes the instructions on the data and returns the results to the computer system's main memory. Within the CPU, the specialized blocks of transistors represented in Figure 6 perform separate functions. The control unit within the CPU performs program flow control and farms out arithmetic computations to one or more arithmetic-logic units (ALUs). The primary ALUs in typical CPUs are often well-optimized for integer arithmetic, with additional less efficient and less costly ALUs performing floating-point arithmetic operations. Instructions and data are held in cache memory within the CPU while the instructions and data are being worked upon, since the access time to cache memory is much shorter than the access time to the computer system's main memory. All operations in a CPU are performed synchronously with a system clock; as a result, the clock speed can be a limiting value of the CPU's performance. Finally, a single computer system may contain one or multiple CPUs.

Flynn's Taxonomy identifies four classes of computational parallelism as shown in Figure 7 [8]. A computer system that runs without parallelism can be described as Single Instruction, Single Data-stream (SISD), in which one instruction or operation is performed

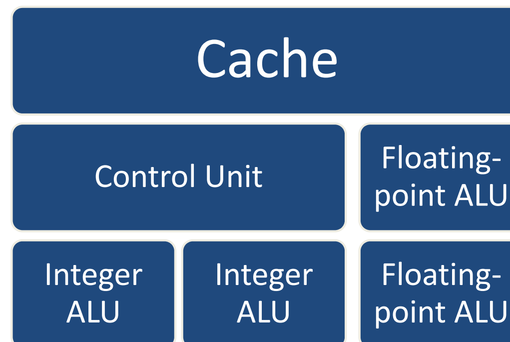

Figure 6: A general-purpose CPU block diagram.

at a time, and on one piece of data at a time. Conventional single-CPU or uniprocessor computers were historically SISD in nature, though with modern architectures this is no longer strictly true. A parallel computer system operates on more than one instruction stream simultaneously, more than one data stream simultaneously, or both more than one instruction stream and more than one data stream simultaneously. A vector processor, which is designed to perform the same mathematical operation on multiple data elements simultaneously, is one type of Single Instruction, Multiple Data-stream (SIMD) computer system. While a few Multiple Instruction, Single Data-stream (MISD) machines exist, they do not suit the majority of problems, and would require that only one piece of data to be operated on at a time, while there might be many instructions. A Multiple Instruction, Multiple Data-stream (MIMD) computer system can accommodate partitioning a problem out in a manner that allows sub-problems to be run simultaneously without either the instruction stream or the data stream needing to be synchronized. While for a period in the 1980s and 1990s SIMD vector processors were dominant type of supercomputers, MIMD parallelism is the most common type of parallel computing in use today.

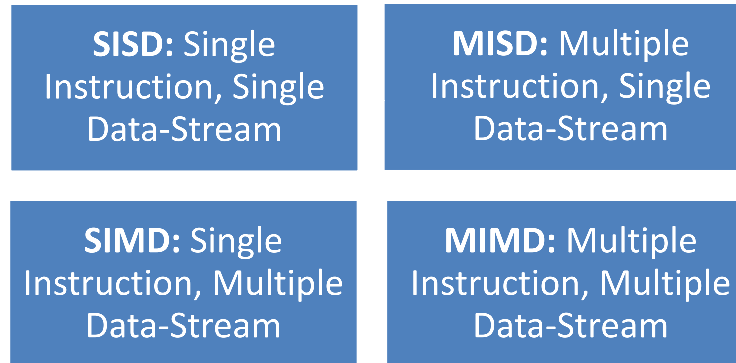

Figure 7: Four classes of parallelism in computers, which together are known as Flynn's Taxonomy [8].

3.2 Parallel Computing with GPGPUs

Graphical Processing Units (GPUs) have been used to aid CPUs in producing sequences of images for more than four decades [54][55]. Over those four decades CPUS and GPUs have followed divergent development paths, since CPUs are most often designed to be as general-purpose as possible, while GPUs have been carefully refined over time for a very small set of operations. GPUs were not originally intended for anything other than a very narrow task set of graphical rendering processes. As a result, the architecture of CPUs and GPUS are now very different from each other and require different programming approaches [56]. It is only in the last few years that it has been possible, with the advent of GPGPU technology, to use the unique capabilities of GPUs for calculations other than graphics processing without first translating the desired mathematical operations into graphics operations [57].

A GPU is a dedicated unit for graphics rendering that acts in conjunction with the CPU

on a computer, the GPU having its own on-chip floating point arithmetic-logic units, memory, and ability to run hundreds or thousands of computational threads in parallel. Program control is retained by the CPU, with specialized graphics instructions sent to the GPU for floating-point computation. In a typical modern CPU, roughly half of the main block is devoted to cache, one quarter to the control unit, and one quarter to the arithmetic logic units, resulting in only a quarter of the transistors being available to do numerical computation. On a GPU, most of the transistors are devoted to arithmetic logic operations and available to do numerical computation as shown in Figure 8; using a GPU to accelerate computation more than quadruples the transistor area devoted to floating-point computation. While the earliest GPUs in the 1960's were developed for Computer-Aided Design (CAD) and flight simulation applications [54], large markets for personal gaming computers with advanced graphics capability have more recently driven development of moderately-priced graphics acceleration boards for commodity computers. Gaming GPUs have developed rapidly in the last ten years, primarily by increasing the number of computational cores on a GPU board [58], while since 2002, due to microchip power constraints and process variations, Intel, AMD, and IBM are no longer scaling CPU clock frequencies as rapidly as they had been in production hardware released to the general public, though other more advanced hardware may be produced for release to the government only. As a result, gaming GPUs have become very powerful floating-point co-processors to desktop CPUs, although highly specialized for graphics operations.

Prior to 2007, all use of the GPU had to be done through the narrowly-focused graphics Applications Programming Interface (API) which was designed in both the physical

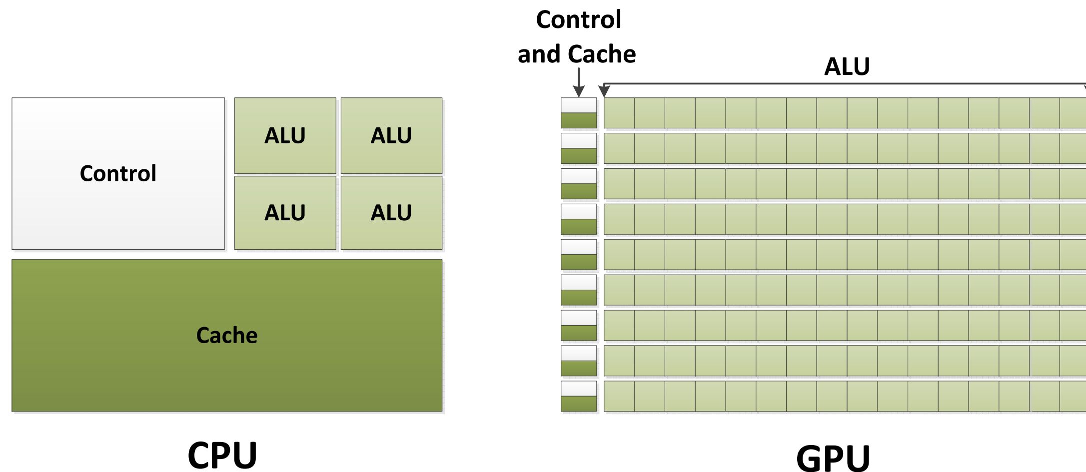

Figure 8: A CPU augmented by a GPU, showing transistor area dedicated to each function [9].

hardware of the GPU and the software to be optimized for graphics rendering only. Starting with the G80 series of chips/boards, Nvidia (a GPU manufacturer) added a general purpose API to their GPUs by modifying both the hardware and the software layers [10]. General-purpose GPU, or GPGPU, interfaces make it much less difficult to attempt to use the floating-point operations power of GPUs for purposes other than graphics rendering. A General-Purpose Graphical Processing Unit (GPGPU) is a computer subsystem chip or card that serves as a general-purpose floating-point accelerator for desktop computers. Some GPGPUs retain their primary structure and command set geared toward graphics rendering with the addition of general-purpose capabilities; others, though still classified as GPGPUs, have appeared on the market with the specialized graphics commands and structures removed, providing a device suitable for floating-point calculation problems that do not require graphical processing at any stage. GPGPU manufacture and programming are both rapidly developing and rapidly changing fields, with the appearance of new GPGPU

boards at frequent intervals and the emergence of GPGPU API standards [57].

Graphical rendering involves a sequence of processes performed on an image, with each process being performed on numerous individual points or pixels [55]. As a result, each imaging process is a problem suited to SIMD parallelism. Further, parts of the image can proceed on to the next process while other parts of the image are still in a previous process, which lends itself well to MIMD parallelism. In total, graphics rendering can be regarded as a set of SIMD problems contained within a MIMD problem. It would not be surprising, then, if over the past 40 years of development, graphical processing units came to be organized as a set of MIMD processors, each of which contains a set of SIMD floating-point processors, and this is exactly what has happened. The historical supercomputers with the most similar architecture were the early Connection Machines line from Thinking Machines in the late 1980s and early 1990s [59]. The earliest Connection Machines sold for $5 million, and a few dozen were sold in total, each of which constituted a unique prototype in its manufacture. An Nvidia GeForce 680 GTX GPGPU card retails for under $500, and the CUDA-endabled GeForce series of which it is a part has passed 6 million in sales by 2009, so that each such card is an easily-replaceable piece of inexpensive and widely-available hardware.

Differences between a Connection Machine and a GPGPU include not only price, power and cooling requirements, and startling differences in physical size; another key difference is that a GPGPU is only a floating-point accelerator to a CPU and program control resides in the CPU. But both the Connection Machines and GPGPUs represent the relatively unusual general-purpose floating point parallelism of a SIMD-architecture within a MIMD architecture. The long history of dedicated GPU development combined

with demand for gaming graphics and continuing speedup of computer technology over the last two decades has brought SIMD-within-MIMD parallel hardware to be available on a PC board, from its last incarnation which filled a large room.

3.3 A GPGPU as a Floating-Point Computation Accelerator

GPGPU chips are designed for a different set of primary tasks from those of modern desktop CPUs, and as a result have substantially different architecture and present different programming challenges. Current GPGPUs consist of an array of arithmetic-logic units within each of a set of multiprocessors as shown in Figure 9.

A typical GPGPU chip contains a set of highly-threaded Streaming MultiProcessors (SMPs) and on-chip memory that is available to all the multiprocessors. Each of those multiprocessors will contain a small instruction unit analogous to the control unit in a CPU, memory analogous to the cache on a CPU, multiple floating-point multiply/add units each with their own registers, and a few special-function units for advanced arithmetic. The floating-point units or processor cores of a single multiprocessor also share access to several types of fast memory. In addition to the components shown in Figure 9, Nvidia GPGPU models issued after 2010 have additional layers of on-chip cache memory to help streamline memory operations to the floating-point units. One possible GPGPU configuration, eight multiprocessors each with 192 floating-point units would give a total of 1536 floating-point units that can proceed in parallel [10][60][61].

A programming model determined by the hardware, firmware, and software interfaces

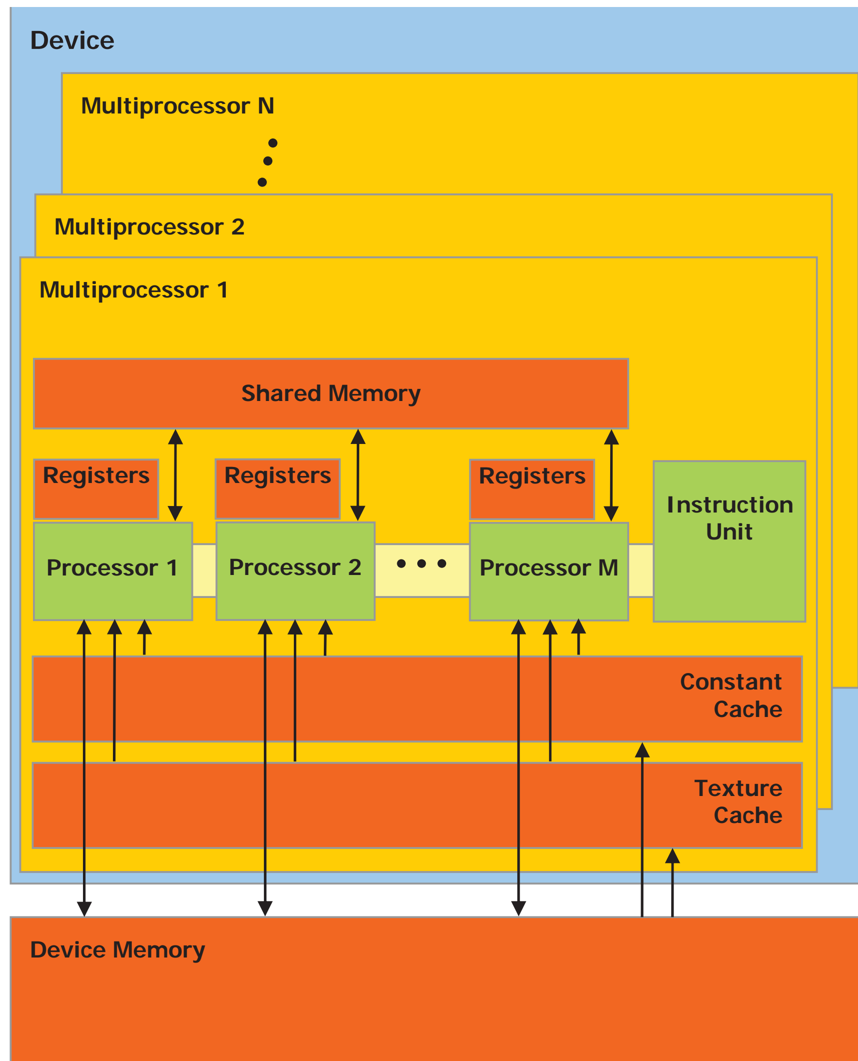

Figure 9: A layout of a typical GPGPGU [10].

combined maps computation to the GPGPU hardware. Program control is executed on the computer system's CPU, from which GPGPU kernels can be launched to send computation to the GPGPU. Instructions for transporting data to and from the GPGPU from the computer system main memory are also issued by the CPU. GPGPU kernel function calls launch blocks of computation threads which are then mapped to floating-point units on the GPGPU through a somewhat complicated hierarchy. In brief, each thread block is mapped to a single SMP, and each SMP breaks up its thread blocks into warps of threads. The individual threads in a warp each execute on a different floating-point unit within the SMP, but the they do so in SIMD fashion – all threads in a warp execute the same instructions in lockstep with each other, but operate on different data. Each SMP executes a SIMD-within-MIMD programming model, meaning that while at every clock cycle each floating-point unit of one warp of threads within an SMP executes the same instruction as the others but on different data, different warps of threads may execute different instruction streams. Each floating-point unit assigned to a thread warp runs a thread identical to that running on the other floating-point units assigned to the same thread warp, but with its own register state and instruction address [57][10][60][61][62][63][64].

As a result, program branching within a warp will result in some threads in the warp being held idle while the branch is executed by the affected threads, so it is desirable to have no branch instructions that differ for the threads in a warp. In addition, the warp size for a given GPGPU is a constant – all thread warps contain the same number of threads. This means that a thread block makes the most use of GPGPU resources if the thread block size is an integer multiple of the warp size for that GPGPU.

The programming approaches in most common use in scientific computing research

laboratories before the advent of GPGPU technology are not well-suited to working with GPGPUs. Most parallel scientific computing was done through interfaces such as MPI that isolate the programmer from being concerned with low-level memory management and subprocessor control. Such tools are just beginning to be developed for GPGPU programming, and the requirements stemming from the physical layout of GPGPUs are unfamiliar to many scientific programmers. One key facet that takes considerable practice to implement is extreme simplicity of algorithms; the GPGPU excels at rapidly performing tens of thousands or more extremely similar but very simple computations. CPUs are often superior at smaller numbers of complex computations, and most problems that are well-suited to GPGPUs will still contain these. Differentiating which computations should be reserved to the GPGPU from which computations should be delegated to the CPU is a critical step in developing programming methods for leveraging GPGPU hardware. Memory management at a very low level also becomes critical; without it the bus between the GPGPU and main computer system memory becomes a bottleneck. Coalescing any data movements into contiguous blocks and moving computational threads rather than the data wherever possible are both necessary for fast results. Simple conditional statements, rather than those with complex calculations in the condition evaluated, also prevent stalling the program momentarily. Until relatively recently, the requirements of memory management for GPGPUs made assembly programming a useful and almost necessary background skill, and it was most effective to treat the high-level language compiler as merely a mask to the assembler and write code for direct translation rather than compilation. More recent versions of software development interface packages for GPGPUs have features that allow high-level language programming without needing to resort to tweaking the assembly

code.

An important limitation on Nvidia GPGPUs is the reduction in 64-bit native or double-precision floating-point support compared to support for single-precision floating-point arithmetic [10][60][61]. GPUs intended for graphics processing only had no need for native 64-bit operations since 32-bit color in graphics is fully supported by 32-bit native or single-precision floating point operations. However, 64-bit native or double-precision floating point precision is desired for most scientific computing problems. Until 2009, GPGPUs with native double-precision floating-point precision were not available, though some GPGPUs did provide software emulation for double-precision calculations, which necessarily comes with a substantial penalty in performance. Further, most GPGPUs have either been partially or not at all compliant with the Institute of Electrical and Electronics Engineers (IEEE) 754 [65] standard for floating-point arithmetic operations [10][60][61]. Double-precision floating point support continues, to be a problem; it may be available only in a form of emulation, and if any form of double-precision support is available for a GPGPU, it is likely to be orders of magnitude slower than single-precision. Problems that can be well-conditioned enough to run with single-precision arithmetic are much easier to map to GPGPUs in a way that produces significant gains in speed of computation.

The GPGPU architecture of a set of multiprocessors, each containing a set of SIMD processors, means that a single multiprocessor on the GPGPU is well-suited to a problem that consists of a set of SIMD subproblems – a set of subproblems with substantial independence from one another, but each subproblem amenable to executing the same instructions on many different pieces of data. While this architecture has substantial advantages for suitable problems, it present challenges unfamiliar to many parallel programmers

and requires far more careful handling of memory and other hardware issues than current supercomputers tend to require of scientific computing programmers.

3.4 Current GPGPU Technology

2007 saw the release of the first GPGPUs, which added to GPUs a general purpose API in addition to the graphics-specialized API by modifying both the programming interface and the device itself. In February of 2007, Nvidia released the G80 series of chips/boards. Advanced Micro Devices (AMD) released the Radeon R580 series in December of 2007. The floating-point operational speed claimed by GPGPU manufacturers has been increasing according to Moore's Law for several years [9][11]. While the shortage of history associated with GPGPU technology means that programming GPGPUs is still tied closely to the details of the hardware itself and effective methods abstracted away from hardware considerations are in the early stages of development, programming standards for GPGPUs and standardized toolsets for tasks such as linear algebra have been steadily increasing in capabilities.

3.4.1 Nvidia

Nvidia currently offers three lines of cards with GPGPU technology; the GeForce, the Tesla, and the Quadro [57][10][60][61]. The GeForce line of GPUs was originally developed for computer gaming video optimization, and the G80 series of the GeForce line became the first GPGPUs. The Quadro line was originally developed to enhance video for Computer-Aided Design (CAD) workstations when Linux began to supplant SGI's Irix

as the operating system of choice for CAD work. With the success of the G80 GPGPUs, GPGPU capability was added into later Quadro cards as well. Additional Quadro GPUs have been developed for finance and trading applications. The Tesla line is intended for general-purpose floating-point computation only; Tesla cards do not contain the interface to the specialized video commands and do not serve as video cards. Nvidia claims its Giga-Floating Point Operations Per Second (GFLOPS) rates have been scaling roughly according to Moores Law since 2000 as shown in Figure 10. Nvidia's developmental focus appears to be on increasing the maximum theoretical memory bandwidth, as shown in Figure 11, and in increasing the number of floating-point arithmetic-logic units per GPGPU.

Nvidia GPGPUs have been compliant with IEEE 754 [65] in most respects, and the cases where full compliance is not supported are carefully documented [10][60][61]. For double precision there are no deviations from the IEEE 754 standard. Of greatest concern is that the precision of division is lower than single-precision.

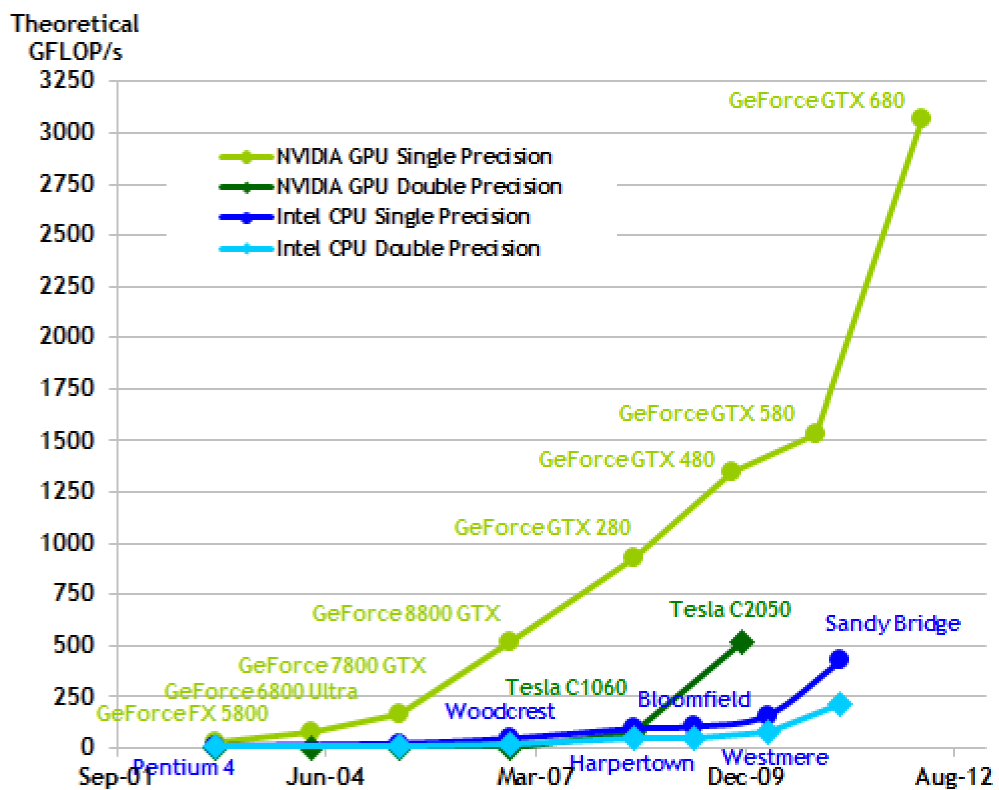

Figure 10: Growth of Nvidia GFLOPs rate scaled roughly according to Moores Law through 2011 [9].

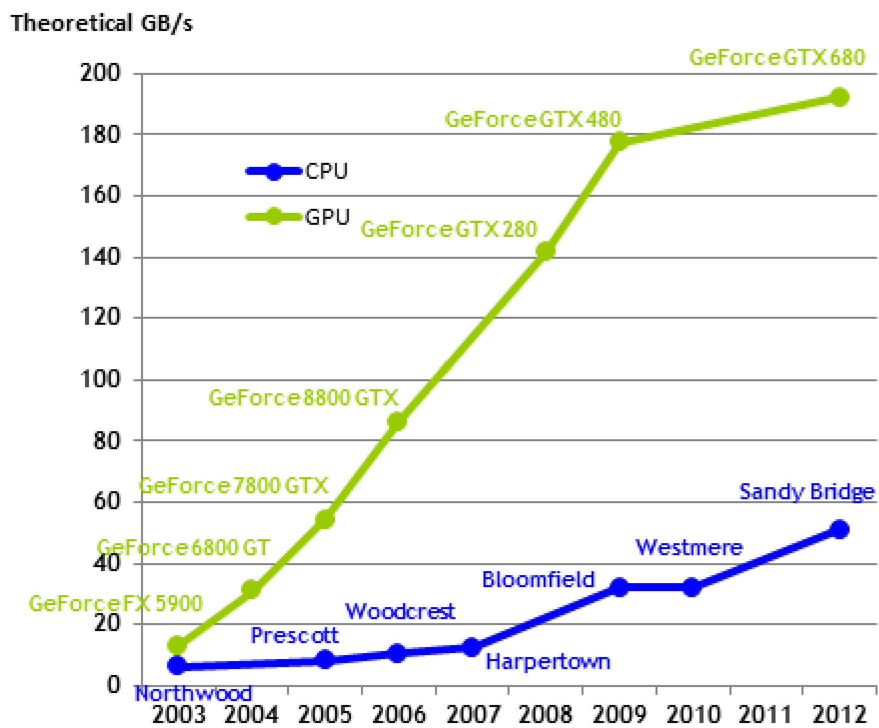

Figure 11: Nvidia has been aggressively increasing their theoretical maximum memory bandwidth [9].

3.4.2 AMD/ATI

AMD/ATI currently offers two lines of cards with GPGPU capability; the ATI Radeon series and the AMD Fire series. AMD purchased ATI, a graphics processing unit manufacturer, in 2006, and the ATI Radeon HD line of cards was developed from ATI computer gaming video enhancement technology. The development of the AMD Firestream line and the subsequent Fire series has been focused on general-purpose high-performance computing. The Firestream 9170, released in November 2007, became the first GPGPU with 64-bit floating point capability and had 320 floating-point arithmetic-logic units. AMD seems to be focusing on intensifying the power of each processing core, as opposed to Nvidia, which is focusing on adding more cores on a single board. AMD/ATI claims its GFLOPs rates have been scaling roughly according to Moores Law since 2005 as shown in Figure 12 [11].

AMD GPGPUs have been nearly compliant with IEEE 754 [65], and the deviations from full compliance are carefully documented [11]. While AMD produced the first GPGPU with native double-precision support, they do not yet offer a version fully compliant with IEEE 754. As with Nvidia, of greatest concern is that the precision of division is lower than single-precision.

3.4.3 Programming Models

Programming models for GPGPUs are still closely tied both to application and the specific GPGPU hardware. In order to write general-purpose code for GPU computing, there needs to be some way to tie a high-level programming language to the general-purpose

Figure 12: AMD/ATI GFLOPs rates have been scaling roughly according to Moores Law since 2005 [11].

floating-point interface on the hardware itself. The GPGPU manufacturers, Nvidia and AMD, provide drivers and tools to perform this function, with a variety of possibilities for the high-level language that sits on top. Each has a most common development path, however, that has been most widely used in the short time these tools have been available, and thus represent the greatest amount of literature available. Since Nvidia GPGPUs were used to develop the methods detailed in this thesis, the remainder of this chapter and thesis will focus on Nvidia GPGPUs and associated programming models. For Nvidia cards, the Compute Unified Device Architecture (CUDA) layer provides the interface between hardware and the high-level language, and C has been the language of choice for most developers. Since a single software layer between the GPGPU hardware and high-level

programming language is the most efficient in terms of performance, and since the newness of GPGPUs makes impressive performance numbers of primary concern to the hardware vendors, software interfaces and the programming models they support are closely tailored to the fine details of the hardware of the vendors that provide them without regard for portability. This may change as the technology matures.

3.4.4 Programming Standards

High-performance computing requires a substantial amount of developer time and investment; a natural concern is that code developed should not be dependent, if possible, on a specific hardware product line. While there will always remain substantial trade-offs between portability and performance, portability of code protects the investment against hardware product lines dying out, and enables less expensive hardware upgrades by widening the range of choices available. Adherence to a widely-accepted programming standard for GPGPU programming would protect the investment in code development, ensuring that the code developed would run on all hardware that supports the standard. A more generalized standard for heterogeneous floating-point computing floating-point computing using multiple machines/boards of different kinds would provide even more protection.

Specification of a standard represents the first in several stages of a standards development; the second stage involves enough vendors offering products that support the standard for it to be usable in practice, while the third stage involves widespread usage of the standard so that it becomes a standard in practice as well as in definition; this can be either a very rapid or a slow process. A number of hardware developers, including Nvidia, Intel, AMD, and Apple, formed the Khronos Compute Working Group in June of 2008 to create

the OpenCL standards. OpenCL 1.2 is an open standard (non-proprietary) for heterogeneous systems aiming to standardize general-purpose parallel programming practices, and a set of conformance tests are available [66]. Vendor support for OpenCL includes AMD, Nvidia, and IBM as part of its XL compilers. Additionally, there is some investigation by Field-Programmable Gate-Array (FPGA) vendors into tools to allow OpenCL to be used for FPGA computing. However, some of the hardware vendors layer OpenCL on top of their own drivers instead of implementing it directly in a single layer; this has negative performance impacts. Because hardware vendors may support OpenCL by adding an extra layer with the result that adhering to the standard loses some performance, adoption of the standard may be slower than it otherwise would be. Further, the OpenCL standard is aimed at expert programmers, more so than the existing toolkits provided by GPGPU manufacturers; benefits to smaller application developers will probably come through toolsets published by the expert community, adding another possible delay in widespread adoption of the standard. Reasons for applications developers to adhere to the standard involve engineering benefits, depending on the application, and marketing benefits if the customer base desires a product that adheres to the standard.

Microsoft DirectCompute is an API that supports general-purpose computing on graphics processing units on Microsoft Windows Vista, Windows 7 and Windows 8, and is intended to serves as a programming standard insofar as it works with multiple hardware vendors. DirectCompute was initially released with the DirectX 11 API but now runs on both DirectX 10 and DirectX 11 graphics processing units.

3.5 Chapter Summary

While currently there are two major manufacturers and several product lines for commodity GPGPU hardware, the methods details in this thesis were developed using Nvidia GPGPUs and associated toolsets and programming interface models. The manufacturers continue to produce new boards and new drivers, and a drive towards at least one programming standard is well underway. While GPGPU programming models, methods and standards are still maturing, requiring close attention to the physical layout in programming a new GPGPU application, the speed potential makes GPGPU programming a worthy option for problems that are well-suited to the SIMD-within MIMD architecture of GPGPUs. GPGPUs to accelerate floating-point computation can be used as part of the inexpensive commodity hardware discussed in Chapter 2 as being desirable for power systems contingency analysis.

Chapter 4

A Novel GPGPU Method for Power Systems Contingency Analysis

As discussed in Chapter 2, power systems contingency analysis involves performing a power flow calculation on each of a list of contingencies, where one contingency represents one or more elements out of service in the power system. This chapter discusses a multi-element approach to mapping that problem to a GPGPU-accelerated computer system. The method outlined here demonstrates computing the set of (N-1) contingencies. The set of (N-2) contingencies can be computed by using the solutions of the set of (N-1) contingencies as the base cases.

This chapter 1) describes the design criteria for the method proposed in this chapter, 2) gives an outline of the basic Fast Decoupled Power Flow (FDPF) algorithm, 3) shows an alteration of the FDPF to perform more of the computation in rectangular notation, 4) derives the current equation updates for the set of (N-1) contingencies, 5) develops using the Matrix Inversion Lemma to avoid computing matrix inversions for the (N-1) contingencies, 6) describes one proposed FPDF contingency algorithm for a single contingency that uses a fast approximation, and 7) describes a fuller contingency algorithm that might converge in fewer iterations and serve better for (N-x) cases, but has less ideal properties

for the purposes here. The chapter concludes with a summary.

4.1 The Design Criteria for the Proposed Method

When this project was started in early 2008, CUDA and devices capable of supporting it had only been available for seven months. GPGPU programming was in its infancy and the hardware and firmware were in early stages of development. Based on the author's previous work with emerging computational hardware, firmware, and software, it was estimated that a mature set of software development tools, hardware, and firmware would likely be in place in the 2014-2015 timeframe and that developing code to use CUDA-enabled GPGPUs would be a very different process than in 2008. Selecting methods to pursue thus became a matter of balancing two desired traits:

1. Methods that would work within the many and substantial limitations of the 2008 technology.

2. Methods that would still be of interest when the technology matured.

4.1.1 Characteristics and Limitations of GPGPU Technology

While some of the limitations of GPGPU technology discussed here have been improved with new versions of the technology, others are fundamental to the design of GPGPUs.

A GPGPU is Large, but Slow

A GPGPU is not a computationally fast device, only a computationally large one. It contains a great many individual floating-point arithmetic units, but those units are individually slow compared to arithmetic units on conventional CPUs. A GPGPU can achieve a high GFLOPS rate by the sheer number of floating-point computations it can perform simultaneously, not by doing any of them quickly. In order to achieve a high GFLOPS rate, the arithmetic units on the GPGPU must be kept constantly busy [57][10][60][61][62][63] [64]. For this to happen, several factors have to be considered:

- There must be a large number of kernel threads launched at any given moment, in the thousands if possible. The GPGPU needs to pipeline a new block of threads to any part of the GPGPU that is not busy. To achieve this it needs a high number of thread blocks launched and waiting in the pipeline.

- Computation assigned to a thread block must be must be arranged to suit the architecture of GPGPU and its execution model. Warps of threads within thread blocks perform computation in SIMD parallelism, with each thread in a warp performing the same set of instructions on different data. As a result, the problem needs to be set up so that there are chunks of SIMD parallelism sized to at least the warp size of the GPGPU.

- Thread blocks need relative independence from one another. The GPGPU pipelines new blocks of threads to SMPs within the GPGPU, and the order in which pipelined blocks are assigned and executed is not easily predictable unless steps are taken to synchronize, and therefore slow down, parts of the pipeline. As a result, thread

blocks that are likely to execute on the same time on the GPGPU need to have independence from one another as much as can be arranged. This means that any thread block ideally should not have to wait for data that is still being computed by another thread block.

- Data transfer between the computer system main memory and the GPGPU needs to be carefully managed. The data needed by those thread blocks that are currently executing needs to be present on the GPGPU, having been transported from the computer system main memory by CPU commands. Since there is limited memory space available on the GPGPU and loading data from the main computer system memory is a comparatively slow process, loading data from the main computer system memory ideally should be interleaved with computations taking place on the device, so that by the time new data is needed by thread blocks ready to execute, that data is already in place. Unfortunately, concurrent kernel execution and data transfer were not possible on the early GPGPUs, and the computation simply had to wait while data transfers were taking place. Whether interleaving data transfer with computation or not, minimizing data transfer between the main computer system memory and the GPGPU can heavily impact performance.

Single-Precision Arithmetic

At the start of this project in 2008, the GPGPUs available could perform not perform double-precision arithmetic but rather single-precision arithmetic only. Double-precision arithmetic was not introduced until the Fermi architecture in 2010. Using double-precision arithmetic halves the speed at which a computation would run on Nvidia GPGPUs when

compared with single-precision arithmetic [10][60][61].

Using single-precision arithmetic has the added advantage that single-precision values take up less storage space on the GPGPU than double-precision values.

Dubious Arithmetic

GPGPU floating-point units are designed for very rudimentary arithmetic operations. Operations of any complexity are sent to special function units on the GPGPU, which are slower than the general GPGPU floating-point units. In the earliest versions, the special function units had only a few operations such as sine, cosine, reciprocal, and square root. The special function units have increased in number and in features in newer versions of the technology, however it is still not recommended to use them without testing whether a numerical approximation on the main GPGPU floating point units might not outperform the special function units for the application in question [10][60][61].

GPGPU arithmetic has a history of exceptions to compliance with the IEEE Standard 754 for floating point operations [10][60][61]. The exceptions have been gradually reduced with newer versions of the technology. The exceptions could affect the convergence of iterative solvers.

The first CUDA-enabled GPGPUs lack of atomic operations means an increased number of roundings [10]. For example, an atomic multiply-add would perform the operation $a = a + b * c$ as a single operation with only one rounding as opposed to performing it with two operations with rounding after each.

Rudimentary Control Logic

Primary program control is executed by the computer system's CPU, and the control units on GPGPUs were not originally intended to handle more than very basic program control (as opposed to thread, warp, and thread block scheduling) [10][60][61]. Logic and branching statements within the threads of a warp can lead to threads sitting idle while only some threads in the warp execute the logic or branch, which can heavily impact performance. While the Nvidia Kepler architecture introduced in 2012 has some improvements to control capabilities on the GPGPU, the presumption of program control being handled on the computer system CPU is still fundamental to the basic GPGPU design [61].

Lack of Sparse Matrix Solvers

When this project began in early 2008, GPGPU basic linear algebra solvers of any kind were in very early stages, employing dense arithmetic only. The routines available had some matrix-vector operations, and fewer matrix-matrix operations such as multiplying a matrix by a constant.

In late 2009, the author spoke to an NVIDIA application developer about whether there were GPGPU sparse matrix solvers in existence. There were proprietary sparse solvers in development at that time, and the NVIDIA developer surmised that there might be a few routines buried somewhere at various national labs, but there was nothing generally available.

Proprietary commercial solvers are available in 2013 and claim impressive results, but the details of their implementations are not generally available. However, research into sparse matrix methods has meant some very good algorithms have become available since

this project began, though only recently.

4.1.2 Design Criteria Chosen Due to Characteristics and Limitations of GPGPU Technology

The GPGPU characteristics and limitations discussed above drove the design criteria for the methods described in this thesis.

Power Flow Algorithm: FDPF

The single-precision arithmetic that is native to GPGPUs drove the decision to work with variants of the FDPF, since the FDPF matrices are well-conditioned and the method is robust under single precision. The FDPF has additional advantages in that it is amenable to the methods for separation of calculations and substitution described in this chapter and the next.

Large Problem Size

The methods developed here are designed for power systems with thousands to tens of thousands of lines. As a general rule, GPGPUs show the greatest speedup over CPU architectures when thousands of threads are launched at any one time [57][10][60][61][62][63][64], and small systems and small problems usually cannot provide enough computation to keep the thread count high enough to keep the floating-point units on the GPGPU occupied.

Balancing Thread Count With Instructions Per Thread

While maintaining high thread count and keeping the maximum number of GPGPU floating-point units occupied with computational threads are generally considered fundamental GPGPU programming practices [57][61][62][63], higher occupancy of floating-point units by threads and/or high thread count do not necessarily translate to the highest performance. The overhead in creating GPGPU threads is low, but not non-existent, so that launching new threads instead of combining computations within threads can degrade performance [67].

Vector Parallelism

The computations for (N-1) contingency analysis are combined in a fashion to create a high-degree of vector or SIMD parallelism to facilitate uniform threads within warps and flexibility in determining thread block size according to the specific GPGPU and computer system in use.

Reduction and Re-Mapping of Sparse Matrix Operations

Because of lack of GPGPU sparse matrix solvers until very recently, the proposed method keeps sparse matrix operations to a minimum. Matrix inversions are avoided entirely except in the pre-calculation stage where they can be performed using established sparse solvers for CPUs.

The multiplication of a sparse matrix by a dense vector is fundamental to power flow algorithms including the FDPF. While the method detailed in this chapter does not specify

which of a number of possible approaches must be used for multiplication of a sparse matrix by a dense vector, this method, by slicing across contingencies, re-maps the computations for contingency analysis from a series of sparse-matrix-dense-vector multiplications separated by intervening computations and data movements to the much more efficient problem of a sparse matrix multiplied by a block of dense vectors larger than the matrix itself.

Reduced Transporting of Large Sparse Matrices

Sparse matrices take up more space per unit of actual data than dense matrices with a few exceptions. If they are stored as dense matrices, they contain mostly zeros. If they are stored as some kind of ordered list paired with index terms, the index terms take up space in addition to the data points. One exception that is useful for power systems computations is diagonal matrices, which can be stored as a vector and indexed as such. Since the method proposed here is to be designed for very large systems and the sparse matrices involved are correspondingly large, one criterion is to reduce where possible such matrices must be transported from the computer system main memory to the GPGPGU or from the GPGPU to the computer system main memory.

Flexibility of Implementation

The method described in this thesis is fairly flexible with respect to a number of implementation details such as thread block size and finer details of data movement to allow for mapping to GPGPUs with differing sets of specifications or to allow mapping to multiple GPGPUs.

Minimizing GPGPU Control Logic

Program branching and control is reserved to the computer system CPU.

Simple Arithmetic

The proposed method focuses on minimizing advanced arithmetic of any form, since the GPU's strengths are in the simplest floating-point arithmetic operations. Approximations are used for trigonometric functions, and reciprocals are avoided where possible.

4.1.3 Methods Chosen

The methods chosen based on the above criteria are expanded upon in the remainder of this chapter, and revolve around setting up the computation of the set of (N-1) contingencies to slice across contingencies to create dense vector operations [68]. Rectangular arithmetic is used for the bus current and power equations so that the $\left[\, \vec{\mathbf{Y}}_{\mathbf{bus}} \,\right]$ matrix [1] can be split into component $\left[\, \mathbf{G}_{\mathbf{bus}} \,\right]$ and $\left[\, \mathbf{B}_{\mathbf{bus}} \,\right]$ matrices. This splits the computation into smaller matrices to facilitate vectorizing the computation across contingencies by leaving more memory space on the device for contingency vectors. A bus current correction method is used to update bus currents, so that the $\left[\, \mathbf{G}_{\mathbf{bus}} \,\right]$ and $\left[\, \mathbf{B}_{\mathbf{bus}} \,\right]$ matrices remain constant and can be used for all (N-1) contingencies. The FDPF is used with the Matrix Inversion Lemma to avoid the need to solve matrix inversions and to facilitate slicing computation across contingencies to replace lengthy series of sparse-matrix-dense-vector multiplications with the much more efficient sparse matrix multiplied by a block of

[1] The $\vec{\ }$ arrow is used to indicate variables, vectors, or matrices that store complex values.

dense vectors larger than the matrix itself.

4.2 The Fast Decoupled Power Flow

This section describes the power flow equations and the basic FDPF algorithm.

4.2.1 Power Flow Equations

The nonlinear network governing equations for any power flow solution method are

$$\vec{\mathbf{S}_\ell} = P_\ell + jQ_\ell = \vec{\mathbf{V}}_\ell \vec{\mathbf{I}}^*_\ell \tag{4.1}$$

where $\vec{\mathbf{I}_\ell}$ is the current injected into bus ℓ and it is equal to

$$\vec{\mathbf{I}_\ell} = \sum_{m=1}^{N} \vec{\mathbf{Y}_{\ell\mathbf{m}}} \vec{\mathbf{V}_{\mathbf{m}}} \tag{4.2}$$

Equation 4.1 in matrix form:

$$\begin{aligned} \left[\ \vec{\mathbf{S}}\ \right] &= \left[\ \mathbf{P}\ \right] + j \left[\ \mathbf{Q}\ \right] \\ &= \left[\ \vec{\mathbf{V}}\ \right] * . \left[\ \vec{\mathbf{I}^*}\ \right] \end{aligned} \tag{4.3}$$

In matrix form, the current equation (4.2) becomes a matrix version of Ohm's Law:

$$\left[\ \vec{\mathbf{I}}\ \right] = \left[\ \vec{\mathbf{Y}_{\mathbf{bus}}}\ \right] \cdot \left[\ \vec{\mathbf{V}}\ \right] \tag{4.4}$$

where:

- $\left[\ \vec{\mathbf{I}}\ \right]$ is an $N_{buses} \times 1$ vector of the net complex currents leaving each bus
- $\left[\ \vec{\mathbf{V}}\ \right]$ is an $N_{buses} \times 1$ vector of the complex voltages at each bus
- $\left[\ \vec{\mathbf{Y_{bus}}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of complex admittance terms

To derive the terms of $\left[\ \vec{\mathbf{Y_{bus}}}\ \right]$, we separate the equation for current into the currents from a bus ℓ to ground and the currents on lines connected to bus ℓ:

$$\vec{\mathbf{I}_\ell} = \vec{\mathbf{V}}_\ell \vec{\mathbf{Y_{\ell G}}} + \sum_{m=1, m\neq\ell}^{N_{bus}} \frac{\vec{\mathbf{V}}_\ell - \vec{\mathbf{V_m}}}{\vec{\mathbf{z_{\ell m}}}} \tag{4.5}$$

where:

- $\vec{\mathbf{Y_{\ell G}}}$ is the sum of the admittances from bus ℓ to ground
- $\vec{\mathbf{V}}_\ell \vec{\mathbf{Y_{\ell G}}}$ is the sum of the currents from bus ℓ to ground
- $\sum_{m=1, m\neq\ell}^{N_{bus}} \frac{\vec{\mathbf{V}}_\ell - \vec{\mathbf{V_m}}}{\vec{\mathbf{z_{\ell m}}}}$ is the sum of the currents on lines connected to bus ℓ

Separating terms for $\vec{\mathbf{V}}_\ell$ and $\vec{\mathbf{V_m}}$

$$\vec{\mathbf{I}_\ell} = \vec{\mathbf{V}}_\ell \left(\vec{\mathbf{Y_{\ell G}}} + \sum_{m=1, m\neq\ell}^{N_{bus}} \frac{1}{\vec{\mathbf{z_{\ell m}}}} \right) - \sum_{m=1, m\neq\ell}^{N_{bus}} \frac{\vec{\mathbf{V_m}}}{\vec{\mathbf{z_{\ell m}}}} \tag{4.6}$$

the terms of $\left[\ \vec{\mathbf{Y_{bus}}}\ \right]$ then become:

$$\vec{\mathbf{Y_{\ell\ell}}} = \vec{\mathbf{Y_{\ell G}}} + \sum_{m=1, m\neq\ell}^{N_{bus}} \frac{1}{\vec{\mathbf{z_{\ell m}}}} \tag{4.7}$$

$$\mathbf{Y}_{\vec{\ell \mathbf{m}}} = \frac{-1}{\mathbf{z}_{\vec{\ell \mathbf{m}}}} \tag{4.8}$$

4.2.2 Inputs to the FPDF

The inputs to the FPDF are:

- $\left[\vec{\mathbf{V}}_{\mathbf{0}} \right]$ is an $N_{buses} \times 1$ vector of the complex voltages at each bus
- $\left[\mathbf{V}_{\mathbf{0}} \right]$ is an $N_{buses} \times 1$ vector of scalar voltage magnitudes at each bus
- $\left[\theta_{\mathbf{0}} \right]$ is an $N_{buses} \times 1$ vector of scalar voltage angles at each bus
- $\left[\mathbf{Y}_{\vec{\mathbf{bus}}} \right]$ is an $N_{buses} \times N_{buses}$ matrix of complex admittance terms
- $\left[\mathbf{B}' \right]$ is an $N_{buses} \times N_{buses}$ Fast Decoupled Power Flow matrix of scalar values
- $\left[\mathbf{B}'' \right]$ is an $N_{buses} \times N_{buses}$ Fast Decoupled Power Flow matrix of scalar values
- $\left[\mathbf{P}_{\mathbf{sched}} \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled active power at each bus
- $\left[\mathbf{Q}_{\mathbf{sched}} \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled reactive power at each bus

The entries for the $\left[\mathbf{B}' \right]$ and $\left[\mathbf{B}'' \right]$ matrices are given in Table 2 and Table 3.

$\mathbf{B}'$	$\mathbf{B}''$
$b'_{\ell m} = -\frac{1}{x_{\ell m}}$	$b''_{\ell m} = -\frac{x_{\ell m}}{r^2_{\ell m} + x^2_{\ell m}}$
$b'_{\ell\ell} = \frac{1}{x_{\ell m}}$	$b''_{\ell\ell} = \left(\frac{x_{\ell m}}{r^2_{\ell m} + x^2_{\ell m}} + b_{cap}\right)$

Table 2: Elements used to form $\mathbf{B}'$ and $\mathbf{B}''$[1].

$\mathbf{B}'$	$\mathbf{B}''$
$B'_{\ell m} = b'_{\ell m}$	$B''_{\ell m} = b''_{\ell m}$
$B'_{\ell\ell} = \sum_{m=1}^{N} b'_{\ell\ell}$	$B''_{\ell\ell} = \sum_{m=1}^{N} b''_{\ell\ell} + b_{shunt_\ell}$
$B'_{\ell\ell} = 10^{+10}$ ℓ=Slack Bus	$B''_{\ell\ell} = 10^{+10}$ ℓ=Slack or PV Bus

Table 3: Formation of $\mathbf{B}'$ and $\mathbf{B}''$ matrices [1].

4.2.3 The $P - \theta$ Iteration:

The $P - \theta$ iteration:

$$\left[\ \vec{\mathbf{I}}\ \right] = \left[\ \vec{\mathbf{Y_{bus}}}\ \right] \cdot \left[\ \vec{\mathbf{V_0}}\ \right] \tag{4.9}$$

$$\left[\ \vec{\mathbf{S}}\ \right] = \left[\ \vec{\mathbf{V_0}}\ \right] * \cdot \left[\ \vec{\mathbf{I^*}}\ \right] \tag{4.10}$$

$$\left[\ \mathbf{P}\ \right] = \Re\left(\left[\ \vec{\mathbf{S}}\ \right]\right) \tag{4.11}$$

$$\left[\ \mathbf{\Delta P}\ \right] = \left[\ \mathbf{P_{sched}}\ \right] - \left[\ \mathbf{P}\ \right] \tag{4.12}$$

$$\Delta P_{max} = \max\left(\left[\ \mathbf{\Delta P}\ \right]\right) \tag{4.13}$$

$$\left[\ \frac{\mathbf{\Delta P}}{\mathbf{V}}\ \right] = \left[\ \mathbf{\Delta P}\ \right] /. \left[\ \mathbf{V}\ \right] \tag{4.14}$$

where $/.$ indicates itemwise division of any vector or matrix (matrices are stored computed as ordered vectors).

Solve for $\left[\ \mathbf{\Delta}\theta\ \right]$:

$$\left[\begin{array}{c}\frac{\mathbf{\Delta P}}{\mathbf{V}}\end{array}\right] = \left[\begin{array}{c}\mathbf{B'}\end{array}\right] \cdot \left[\begin{array}{c}\mathbf{\Delta\theta}\end{array}\right] \tag{4.15}$$

$$\left[\begin{array}{c}\theta_{\mathbf{1}}\end{array}\right] = \left[\begin{array}{c}\theta_{\mathbf{0}}\end{array}\right] + \left[\begin{array}{c}\mathbf{\Delta\theta}\end{array}\right] \tag{4.16}$$

$$\begin{aligned}\left[\begin{array}{c}\vec{\mathbf{V}}_{\mathbf{1/2}}\end{array}\right] = {} & \left[\begin{array}{c}\mathbf{V_0}\end{array}\right] \cdot \cos\left[\begin{array}{c}\theta_{\mathbf{1}}\end{array}\right] \\ & + j\left[\begin{array}{c}\mathbf{V_0}\end{array}\right] \cdot \sin\left[\begin{array}{c}\theta_{\mathbf{1}}\end{array}\right]\end{aligned} \tag{4.17}$$

where $\left[\begin{array}{c}\vec{\mathbf{V}}_{\mathbf{1/2}}\end{array}\right]$ is the vector of bus voltages halfway through the first iteration, when the bus voltage angles have been updated but the bus voltage magnitudes have not. $\left[\begin{array}{c}\vec{\mathbf{V}}_{\mathbf{1}}\end{array}\right]$ is used at the end of the first iteration after the bus voltage magnitudes have been updated as well.

4.2.4 The $Q - V$ Iteration:

The $Q - V$ iteration:

$$\left[\begin{array}{c}\vec{\mathbf{I}}\end{array}\right] = \left[\begin{array}{c}\vec{\mathbf{Y}}_{\mathbf{bus}}\end{array}\right] \cdot \left[\begin{array}{c}\vec{\mathbf{V}}_{\mathbf{1/2}}\end{array}\right] \tag{4.18}$$

$$\left[\begin{array}{c}\vec{\mathbf{S}}\end{array}\right] = \left[\begin{array}{c}\vec{\mathbf{V}}_{\mathbf{1/2}}\end{array}\right] * \cdot \left[\begin{array}{c}\vec{\mathbf{I}}^{*}\end{array}\right] \tag{4.19}$$

$$\begin{bmatrix} \mathbf{Q} \end{bmatrix} = \Im\left(\begin{bmatrix} \vec{\mathbf{S}} \end{bmatrix}\right) \tag{4.20}$$

$$\begin{bmatrix} \mathbf{\Delta Q} \end{bmatrix} = \begin{bmatrix} \mathbf{Q_{sched}} \end{bmatrix} - \begin{bmatrix} \mathbf{Q} \end{bmatrix} \tag{4.21}$$

$$\Delta Q_{max} = \max\left(\begin{bmatrix} \mathbf{\Delta Q} \end{bmatrix}\right) \tag{4.22}$$

$$\begin{bmatrix} \frac{\mathbf{\Delta Q}}{\mathbf{V}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Delta Q} \end{bmatrix} /\cdot \begin{bmatrix} \mathbf{V_{1/2}} \end{bmatrix} \tag{4.23}$$

Solve for $\begin{bmatrix} \mathbf{\Delta V} \end{bmatrix}$:

$$\begin{bmatrix} \frac{\mathbf{\Delta Q}}{\mathbf{V}} \end{bmatrix} = \begin{bmatrix} \mathbf{B''} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{\Delta V} \end{bmatrix} \tag{4.24}$$

$$\begin{bmatrix} \mathbf{V_1} \end{bmatrix} = \begin{bmatrix} \mathbf{V_0} \end{bmatrix} + \begin{bmatrix} \mathbf{\Delta V} \end{bmatrix} \tag{4.25}$$

$$\begin{aligned} \begin{bmatrix} \vec{\mathbf{V_1}} \end{bmatrix} = & \begin{bmatrix} \mathbf{V_1} \end{bmatrix} \cdot \cos\begin{bmatrix} \theta_{\mathbf{1}} \end{bmatrix} \\ & + j\begin{bmatrix} \mathbf{V_1} \end{bmatrix} \cdot \sin\begin{bmatrix} \theta_{\mathbf{1}} \end{bmatrix} \end{aligned} \tag{4.26}$$

If ΔP_{max} and ΔQ_{max} have not converged to within tolerance, iterate by returning to Equation 4.9.

4.3 Rectangular Form for the Current and Power Equations

Computing the current and power equations in rectangular form allows the $\left[\ \vec{\mathbf{Y}_{\mathbf{bus}}}\ \right]$ matrix to be split into component $\left[\ \mathbf{G_{bus}}\ \right]$ and $\left[\ \mathbf{B_{bus}}\ \right]$ matrices. If they are stored as dense matrices, the $\left[\ \mathbf{G_{bus}}\ \right]$ and $\left[\ \mathbf{B_{bus}}\ \right]$ matrices each take up half the space of the $\left[\ \vec{\mathbf{Y}_{\mathbf{bus}}}\ \right]$ matrix. If they are stored as sparse matrices, the $\left[\ \mathbf{G_{bus}}\ \right]$ and $\left[\ \mathbf{B_{bus}}\ \right]$ matrices each take up less space individually than the $\left[\ \vec{\mathbf{Y}_{\mathbf{bus}}}\ \right]$ matrix.

This version of the algorithm requires sine and cosine functions to convert bus voltages from polar to rectangular form. The version in Section 4.2 requires sine and cosine functions to convert the complex powers at each bus from polar to rectangular form.

Splitting the complex vectors and matrices $\left[\ \vec{\mathbf{I}}\ \right]$, $\left[\ \vec{\mathbf{V}_{\mathbf{0}}}\ \right]$, and $\left[\ \vec{\mathbf{Y}_{\mathbf{bus}}}\ \right]$ into rectangular coordinates gives:

$$\left[\ \vec{\mathbf{I}}\ \right] = \left[\ \mathbf{I}'\ \right] + j\left[\ \mathbf{I}''\ \right] \tag{4.27}$$

$$\left[\ \vec{\mathbf{V}}\ \right] = \left[\ \mathbf{V}'\ \right] + j\left[\ \mathbf{V}''\ \right] \tag{4.28}$$

$$\left[\ \vec{\mathbf{Y}_{\mathbf{bus}}}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] + j\left[\ \mathbf{B_{bus}}\ \right] \tag{4.29}$$

As a result, the current equation,

$$\left[\ \vec{\mathbf{I}}\ \right] = \left[\ \vec{\mathbf{Y_{bus}}}\ \right] \cdot \left[\ \vec{\mathbf{V}}\ \right] \tag{4.30}$$

becomes

$$\left[\ \vec{\mathbf{I}}\ \right] = \left(\left[\ \mathbf{G_{bus}}\ \right] + j\left[\ \mathbf{B_{bus}}\ \right]\right) \cdot \left(\left[\ \mathbf{V}'\ \right] + j\left[\ \mathbf{V}''\ \right]\right) \tag{4.31}$$

$$= \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}'\ \right] + j\left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}''\ \right] \tag{4.32}$$

$$+ j\left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}'\ \right] - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}''\ \right] \tag{4.33}$$

giving

$$\left[\ \mathbf{I}'\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}'\ \right] - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}''\ \right] \tag{4.34}$$

$$\left[\ \mathbf{I}''\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}''\ \right] + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}'\ \right] \tag{4.35}$$

The power equation,

$$\left[\ \vec{\mathbf{S}}\ \right] = \left[\ \mathbf{P}\ \right] + j\left[\ \mathbf{Q}\ \right] = \left[\ \vec{\mathbf{V_0}}\ \right] * \cdot \left[\ \vec{\mathbf{I}^*}\ \right] \tag{4.36}$$

becomes

$$\left[\ \vec{\mathbf{S}}\ \right] = \left(\left[\ \mathbf{V}'\ \right] + j\left[\ \mathbf{V}''\ \right]\right) *.\left(\left[\ \mathbf{I}'\ \right] - j\left[\ \mathbf{I}''\ \right]\right) \tag{4.37}$$

$$= \left[\ \mathbf{V}'\ \right] *.\left[\ \mathbf{I}'\ \right] - j\left[\ \mathbf{V}'\ \right] *.\left[\ \mathbf{I}''\ \right] \tag{4.38}$$

$$+ j\left[\ \mathbf{V}''\ \right] *.\left[\ \mathbf{I}'\ \right] + \left[\ \mathbf{V}''\ \right] *.\left[\ \mathbf{I}''\ \right] \tag{4.39}$$

giving

$$\left[\ \mathbf{P}\ \right] = \left[\ \mathbf{V}'\ \right] *.\left[\ \mathbf{I}'\ \right] + \left[\ \mathbf{V}''\ \right] *.\left[\ \mathbf{I}''\ \right] \tag{4.40}$$

$$\left[\ \mathbf{Q}\ \right] = \left[\ \mathbf{V}''\ \right] *.\left[\ \mathbf{I}'\ \right] - \left[\ \mathbf{V}'\ \right] *.\left[\ \mathbf{I}''\ \right] \tag{4.41}$$

where $*.$ indicates the vector inner product operator or itemwise multiplication of any vector or matrix (matrices are stored computed as ordered vectors).

4.3.1 Inputs to the FPDF

The inputs to this form of the FPDF are:

- $\left[\ \mathbf{V_0}\ \right]$ is an $N_{buses} \times 1$ vector of scalar voltage magnitudes at each bus
- $\left[\ \theta_{\mathbf{0}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar voltage angles at each bus
- $\left[\ \mathbf{G_{bus}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of the real parts of the entries in the $\left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right]$ matrix

- $\left[\ \mathbf{B_{bus}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of the imaginary parts of the entries in the $\left[\ \vec{\mathbf{Y_{bus}}}\ \right]$ matrix

- $\left[\ \mathbf{B'}\ \right]$ is an $N_{buses} \times N_{buses}$ Fast Decoupled Power Flow matrix of scalar values

- $\left[\ \mathbf{B''}\ \right]$ is an $N_{buses} \times N_{buses}$ Fast Decoupled Power Flow matrix of scalar values

- $\left[\ \mathbf{P_{sched}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled active power at each bus

- $\left[\ \mathbf{Q_{sched}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled reactive power at each bus

4.3.2 The $P - \theta$ Iteration:

The $P - \theta$ iteration:

$$\left[\ \mathbf{V_0'}\ \right] = \left[\ \mathbf{V_0}\ \right] *. \left(\cos.\left[\ \theta_\mathbf{0}\ \right]\right) \tag{4.42}$$

$$\left[\ \mathbf{V_0''}\ \right] = \left[\ \mathbf{V_0}\ \right] *. \left(\sin.\left[\ \theta_\mathbf{0}\ \right]\right) \tag{4.43}$$

$$\left[\ \mathbf{I'}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V_0'}\ \right] - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V_0''}\ \right] \tag{4.44}$$

$$\left[\ \mathbf{I''}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V_0''}\ \right] + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V_0'}\ \right] \tag{4.45}$$

$$\left[\ \mathbf{P}\ \right] = \left[\ \mathbf{V_0'}\ \right] *\cdot \left[\ \mathbf{I'}\ \right] + \left[\ \mathbf{V_0''}\ \right] *\cdot \left[\ \mathbf{I''}\ \right] \tag{4.46}$$

(4.47)

$$\left[\ \mathbf{\Delta P}\ \right] = \left[\ \mathbf{P_{sched}}\ \right] - \left[\ \mathbf{P}\ \right] \tag{4.48}$$

$$\Delta P_{max} = \max\left(\left[\ \mathbf{\Delta P}\ \right]\right) \tag{4.49}$$

$$\left[\ \frac{\mathbf{\Delta P}}{\mathbf{V}}\ \right] = \left[\ \mathbf{\Delta P}\ \right] /\cdot \left[\ \mathbf{V_0}\ \right] \tag{4.50}$$

Solve for $\left[\ \boldsymbol{\Delta\theta}\ \right]$:

$$\left[\ \frac{\mathbf{\Delta P}}{\mathbf{V}}\ \right] = \left[\ \mathbf{B'}\ \right] \cdot \left[\ \boldsymbol{\Delta\theta}\ \right] \tag{4.51}$$

$$\left[\ \theta_{\mathbf{1}}\ \right] = \left[\ \theta_{\mathbf{0}}\ \right] + \left[\ \boldsymbol{\Delta\theta}\ \right] \tag{4.52}$$

4.3.3 The $Q - V$ Iteration:

The $Q - V$ iteration:

$$\left[\mathbf{V'_{1/2}}\right] = \left[\mathbf{V_0}\right] *. \left(\cos . \left[\theta_\mathbf{1}\right]\right) \tag{4.53}$$

$$\left[\mathbf{V''_{1/2}}\right] = \left[\mathbf{V_0}\right] *. \left(\sin . \left[\theta_\mathbf{1}\right]\right) \tag{4.54}$$

$$\left[\mathbf{I'}\right] = \left[\mathbf{G_{bus}}\right] \cdot \left[\mathbf{V'_{1/2}}\right] - \left[\mathbf{B_{bus}}\right] \cdot \left[\mathbf{V''_{1/2}}\right] \tag{4.55}$$

$$\left[\mathbf{I''}\right] = \left[\mathbf{G_{bus}}\right] \cdot \left[\mathbf{V''_{1/2}}\right] + \left[\mathbf{B_{bus}}\right] \cdot \left[\mathbf{V'_{1/2}}\right] \tag{4.56}$$

$$\left[\mathbf{Q}\right] = \left[\mathbf{V''_{1/2}}\right] *. \left[\mathbf{I'}\right] - \left[\mathbf{V'_{1/2}}\right] *. \left[\mathbf{I''}\right] \tag{4.57}$$

$$\left[\mathbf{\Delta Q}\right] = \left[\mathbf{Q_{sched}}\right] - \left[\mathbf{Q}\right] \tag{4.58}$$

$$\Delta Q_{max} = \max\left(\left[\mathbf{\Delta Q}\right]\right) \tag{4.59}$$

$$\left[\frac{\mathbf{\Delta Q}}{\mathbf{V}}\right] = \left[\mathbf{\Delta Q}\right] /. \left[\mathbf{V_{1/2}}\right] \tag{4.60}$$

Solve for $\left[\mathbf{\Delta V}\right]$:

$$\left[\frac{\mathbf{\Delta Q}}{\mathbf{V}} \right] = \left[\mathbf{B''} \right] \cdot \left[\mathbf{\Delta V} \right] \tag{4.61}$$

$$\left[\mathbf{V_1} \right] = \left[\mathbf{V_0} \right] + \left[\mathbf{\Delta V} \right] \tag{4.62}$$

If ΔP_{max} and ΔQ_{max} have not converged to within tolerance, iterate by returning to Equation 4.42, which now becomes:

$$\left[\mathbf{V_1'} \right] = \left[\mathbf{V_1} \right] *. \left(\cos . \left[\theta_\mathbf{1} \right] \right) \tag{4.63}$$

$$\left[\mathbf{V_1''} \right] = \left[\mathbf{V_1} \right] *. \left(\sin . \left[\theta_\mathbf{1} \right] \right) \tag{4.64}$$

4.4 Contingency Current updates

Rather than correct $\left[\mathbf{G_{bus}} \right]$ and $\left[\mathbf{B_{bus}} \right]$ for each contingency, the base case matrices $\left[\mathbf{G_{bus}} \right]$ and $\left[\mathbf{B_{bus}} \right]$ will be used for an initial calculation followed by a correction to get the current vectors for each contingency. This allows a single version of $\left[\mathbf{G_{bus}} \right]$ and $\left[\mathbf{B_{bus}} \right]$ to reside in GPGPU memory while being accessed by threads performing computations for different contingencies.

4.4.1 Basic derivation

For each contingency n in which a single line from bus ℓ to m is out of service, the basic current equation gives correct currents except for buses ℓ and m:

$$\left[\ \vec{\mathbf{I}}^{(\mathbf{n})}_{\mathbf{temp}}\ \right] = \left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right]\left[\ \vec{\mathbf{V}}^{(\mathbf{n})}\ \right] \tag{4.65}$$

Let $\mathbf{m}^{(\mathbf{n})}$ be a row vector of length $Nbuses$, such that

$$\left[\ \mathbf{m}^{(\mathbf{n})\mathbf{T}}\ \right] \cdot \left[\ \mathbf{m}^{(\mathbf{n})}\ \right] \tag{4.66}$$

gives the line connection matrix for line n. The row vector $\mathbf{m}^{(\mathbf{n})}$ for each $(N-1)$ contingency has two non-zero entries; $\mathbf{m}^{(\mathbf{n})}[\ell] = 1$ and $\mathbf{m}^{(\mathbf{n})}[m] = -1$.

To correct for the current due to the series impedance:

$$\left[\ \vec{\mathbf{Y}}^{(\mathbf{n})}\ \right] = \left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right] - \vec{\mathbf{Y}}_{\ell\mathbf{m}}\left[\ \mathbf{m}^{(\mathbf{n})\mathbf{T}}\mathbf{m}^{(\mathbf{n})}\ \right] \tag{4.67}$$

After which the current equation becomes:

$$\begin{aligned}
\left[\ \vec{\mathbf{I}}^{(\mathbf{n})}\ \right] &= \left[\ \vec{\mathbf{Y}}^{(\mathbf{n})}\ \right] \cdot \left[\ \vec{\mathbf{V}}^{(\mathbf{n})}\ \right] \\
&= [\left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right] - \vec{Y}_{\ell m}\left[\ \mathbf{m}^{(\mathbf{n})\mathbf{T}}\mathbf{m}^{(\mathbf{n})}\ \right]] \cdot \left[\ \vec{\mathbf{V}}^{(\mathbf{n})}\ \right] \\
&= \left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right] \cdot \left[\ \vec{\mathbf{V}}^{(\mathbf{n})}\ \right] - \vec{Y}_{\ell m}\left[\ \mathbf{m}^{(\mathbf{n})\mathbf{T}}\ \right] \cdot \left[\ \mathbf{m}^{(\mathbf{n})}\ \right] \cdot \left[\ \vec{\mathbf{V}}^{(\mathbf{n})}\ \right]
\end{aligned} \tag{4.68}$$

Combining 4.65 and 4.68 gives:

$$\left[\ \vec{\mathbf{I}}^{(\mathbf{n})}\ \right] = \left[\ \mathbf{I}^{(\mathbf{n})}_{\mathbf{temp}}\ \right] - \vec{Y}_{\ell m}\left[\ \mathbf{m}^{(\mathbf{n})\mathbf{T}}\ \right] \cdot \left[\ \mathbf{m}^{(\mathbf{n})}\ \right] \cdot \left[\ \vec{\mathbf{V}}^{(\mathbf{n})}\ \right] \tag{4.69}$$

The value

$$\left[\ \mathbf{m^{(n)}}\ \right]\cdot\left[\ \vec{\mathbf{V^{(n)}}}\ \right]=\vec{V_\ell^{(n)}}-\vec{V_m^{(n)}} \tag{4.70}$$

is a single complex number, and

$$\left[\ \mathbf{m^{(n)T}}\ \right]\cdot\left[\ \mathbf{m^{(n)}}\ \right]\cdot\left[\ \vec{\mathbf{V^{(n)}}}\ \right] \tag{4.71}$$

is a complex column vector with two nonzero entries (the ℓth and mth entries):

$$-\vec{Y_{\ell m}}(\vec{V_\ell^{(n)}}-\vec{V_m^{(n)}})=\ -\vec{I_{\ell m}} \tag{4.72}$$

$$-\vec{Y_{\ell m}}(\vec{V_m^{(n)}}-\vec{V_\ell^{(n)}})=\ -\vec{I_{m\ell}} \tag{4.73}$$

So, to correct the current vector for contingency n:

$$I_\ell^{(n)}=I_{temp,\ell}^{(n)}+\vec{Y_{\ell m}}(\vec{V_\ell^{(n)}}-\vec{V_m^{(n)}}) \tag{4.74}$$

$$I_m^{(n)}=I_{temp,m}^{(n)}+\vec{Y_{\ell m}}(\vec{V_m^{(n)}}-\vec{V_\ell^{(n)}}) \tag{4.75}$$

4.4.2 Rectangular notation

Changing to rectangular notation:

$$\begin{aligned} I_\ell'^{(n)}&=I_{temp,\ell}'^{(n)}+\Re[\vec{Y_{\ell m}}(\vec{V_\ell^{(n)}}-\vec{V_m^{(n)}})] \\ I_\ell''^{(n)}&=I_{temp,\ell}''^{(n)}+\Im[\vec{Y_{\ell m}}(\vec{V_\ell^{(n)}}-\vec{V_m^{(n)}})] \\ I_m'^{(n)}&=I_{temp,m}'^{(n)}+\Re[\vec{Y_{\ell m}}(\vec{V_m^{(n)}}-\vec{V_\ell^{(n)}})] \\ I_m''^{(n)}&={}''^{(n)}_{temp,m}+\Im[\vec{Y_{\ell m}}(\vec{V_m^{(n)}}-\vec{V_\ell^{(n)}})] \end{aligned} \tag{4.76}$$

$$
\begin{aligned}
I_\ell'^{(n)} &= I_{temp,\ell}'^{(n)} + g_{line,\ell m}(V_\ell'^{(n)} - V_m'^{(n)}) - b_{line,\ell m}(V_\ell''^{(n)} - V_m''^{(n)}) \\
I_\ell''^{(n)} &= I_{temp,\ell}''^{(n)} + g_{line,\ell m}(V_\ell''^{(n)} - V_m''^{(n)}) + b_{line,\ell m}(V_\ell'^{(n)} - V_m'^{(n)}) \\
I_m'^{(n)} &= I_{temp,m}'^{(n)} + g_{line,\ell m}(V_m'^{(n)} - V_\ell'^{(n)}) - b_{line,\ell m}(V_m''^{(n)} - V_\ell''^{(n)}) \\
I_m''^{(n)} &= I_{temp,m}''^{(n)} + g_{line,\ell m}(V_m''^{(n)} - V_\ell''^{(n)}) + b_{line,\ell m}(V_m'^{(n)} - V_\ell'^{(n)})
\end{aligned} \tag{4.77}
$$

4.4.3 Adding a shunt corrective term

For simplicity, the derivations thus far have ignored the shunt capacitance of the line model. To include a final correction for the shunt capacitive term:

$$
\begin{aligned}
I_\ell'^{(n)} &= I_{temp,\ell}'^{(n)} - \frac{1}{2} b_{cap,line}(V_\ell''^{(n)} - V_m''^{(n)}) \\
I_\ell''^{(n)} &= I_{temp,\ell}''^{(n)} + \frac{1}{2} b_{cap,line}(V_\ell'^{(n)} - V_m'^{(n)}) \\
I_m'^{(n)} &= I_{temp,m}'^{(n)} - \frac{1}{2} b_{cap,line}(V_m''^{(n)} - V_\ell''^{(n)}) \\
I_m''^{(n)} &= I_{temp,m}''^{(n)} + \frac{1}{2} b_{cap,line}(V_m'^{(n)} - V_\ell'^{(n)})
\end{aligned} \tag{4.78}
$$

To combine Equation 4.77 and Equation 4.78:

$$\begin{aligned}
I_\ell'^{(n)} &= I_{temp,\ell}'^{(n)} + g_{line,\ell m}(V_\ell'^{(n)} - V_m'^{(n)}) - b_{line,\ell m}(V_\ell''^{(n)} - V_m''^{(n)}) \\
&- \frac{1}{2} b_{cap,line,\ell m}(V_\ell''^{(n)} - V_m''^{(n)}) \\
I_\ell''^{(n)} &= I_{temp,\ell}''^{(n)} + g_{line,\ell m}(V_\ell''^{(n)} - V_m''^{(n)}) + b_{line,\ell m}(V_\ell'^{(n)} - V_m'^{(n)}) \\
&+ \frac{1}{2} b_{cap,line,\ell m}(V_\ell'^{(n)} - V_m'^{(n)}) \\
I_m'^{(n)} &= I_{temp,m}'^{(n)} + g_{line,\ell m}(V_m'^{(n)} - V_\ell'^{(n)}) - b_{line,\ell m}(V_m''^{(n)} - V_\ell''^{(n)}) \\
&- \frac{1}{2} b_{cap,line,\ell m}(V_m''^{(n)} - V_\ell''^{(n)}) \\
I_m''^{(n)} &= I_{temp,m}''^{(n)} + g_{line,\ell m}(V_m''^{(n)} - V_\ell''^{(n)}) + b_{line,\ell m}(V_m'^{(n)} - V_\ell'^{(n)}) \\
&+ \frac{1}{2} b_{cap,line,\ell m}(V_m'^{(n)} - V_\ell'^{(n)})
\end{aligned} \tag{4.79}$$

Combining terms:

$$
\begin{aligned}
I_\ell'^{(n)} &= I_{temp,\ell}'^{(n)} + g_{line,\ell m}(V_\ell'^{(n)} - V_m'^{(n)}) \\
&- (b_{line,\ell m} + \frac{1}{2}b_{cap,line,\ell m})(V_\ell''^{(n)} - V_m''^{(n)}) \\
I_\ell''^{(n)} &= I_{temp,\ell}''^{(n)} + g_{line,\ell m}(V_\ell''^{(n)} - V_m''^{(n)}) \\
&+ (b_{line,\ell m} + \frac{1}{2}b_{cap,line,\ell m})(V_\ell'^{(n)} - V_m'^{(n)}) \\
I_m'^{(n)} &= I_{temp,m}'^{(n)} + g_{line,\ell m}(V_m'^{(n)} - V_\ell'^{(n)}) \\
&- (b_{line,\ell m} + \frac{1}{2}b_{cap,line,\ell m})(V_m''^{(n)} - V_\ell''^{(n)}) \\
I_m''^{(n)} &= I_{temp,m}''^{(n)} + g_{line,\ell m}(V_m''^{(n)} - V_\ell''^{(n)}) \\
&+ (b_{line,\ell m} + \frac{1}{2}b_{cap,line,\ell m})(V_m'^{(n)} - V_\ell'^{(n)})
\end{aligned} \tag{4.80}
$$

4.4.4 Expressing the current updates as matrix operations

Noting that the correction for $I_m'^{(n)}$ is the negative of the correction for $I_\ell'^{(n)}$ and that the correction for $I_m''^{(n)}$ is the negative of the correction for $I_\ell''^{(n)}$, in notation for matrix operations, the current correction algorithm becomes:

$$
\left[\ \mathbf{I}'^{(\mathbf{n})}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V_0'^{(n)}}\ \right] - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V_0''^{(n)}}\ \right] \tag{4.81}
$$

$$
\left[\ \mathbf{I}''^{(\mathbf{n})}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V_0''^{(n)}}\ \right] + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V_0'^{(n)}}\ \right] \tag{4.82}
$$

$$\mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust}} = \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right][\ell, m] \cdot \left(\left[\begin{array}{c} \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][m]\right) - \left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right][\ell, m] \cdot \left(\left[\begin{array}{c} \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][m]\right) \tag{4.83}$$

$$\begin{aligned} \left[\begin{array}{c} \mathbf{I}'^{(\mathbf{n})} \end{array}\right][\ell] &= \left[\begin{array}{c} \mathbf{I}'^{(\mathbf{n})} \end{array}\right][\ell] + \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust}} \\ \left[\begin{array}{c} \mathbf{I}'^{(\mathbf{n})} \end{array}\right][m] &= \left[\begin{array}{c} \mathbf{I}'^{(\mathbf{n})} \end{array}\right][m] - \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust}} \end{aligned} \tag{4.84}$$

$$\mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust}} = \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right][\ell, m] \cdot \left(\left[\begin{array}{c} \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][m]\right) + \left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right][\ell, m] \cdot \left(\left[\begin{array}{c} \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}} \end{array}\right][m]\right) \tag{4.85}$$

$$\begin{aligned} \left[\begin{array}{c} \mathbf{I}''^{(\mathbf{n})} \end{array}\right][\ell] &= \left[\begin{array}{c} \mathbf{I}''^{(\mathbf{n})} \end{array}\right][\ell] + \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust}} \\ \left[\begin{array}{c} \mathbf{I}''^{(\mathbf{n})} \end{array}\right][m] &= \left[\begin{array}{c} \mathbf{I}''^{(\mathbf{n})} \end{array}\right][m] - \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust}} \end{aligned} \tag{4.86}$$

where $\frac{1}{2}b_{cap,line,\ell m}$ is included with $b_{line,\ell m}$ in the term $\left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right][\ell, m]$

4.5 The Matrix Inversion Lemma

The Matrix Inversion Lemma and its special cases allow for the inverses of certain combinations of matrices to be computed from component matrices and their inverses. A

special case of the Matrix Inversion Lemma for the inverse of the sum of two matrices assists in running contingency power flows using the FDPF method without recomputing $\left[\begin{array}{c}\mathbf{B}'\end{array}\right]$ and $\left[\begin{array}{c}\mathbf{B}''\end{array}\right]$ and their inverses for each contingency.

4.5.1 Definition

Let:

$$\begin{aligned} \mathbf{A} &\text{ be an } n \times n \text{ invertible matrix} \\ \mathbf{B} &\text{ be an } n \times m \text{ invertible matrix} \\ \mathbf{C} &\text{ be an } m \times m \text{ invertible matrix} \\ \mathbf{D} &\text{ be an } m \times n \text{ invertible matrix} \end{aligned} \tag{4.87}$$

The Matrix Inversion Lemma [69] (also known as the Woodbury matrix identity) states:

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1} \tag{4.88}$$

4.5.2 Sum of Two Matrices Case of the Matrix Inversion Lemma

There are several special-case forms of the Matrix Inversion Lemma, including a form for the inverse of the sum of two matrices, given as follows: In equation 4.88 above, let:

$$\begin{aligned} m &= n, \text{ so that} \\ \mathbf{C} &\text{ is an } nxn \text{ invertible matrix} \end{aligned} \tag{4.89}$$

Further, let

$$\mathbf{B} = \mathbf{I}_{nxn}$$

$$\mathbf{D} = \mathbf{I}_{nxn} \tag{4.90}$$

Then

$$(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1}+\mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1} \tag{4.91}$$

$$\Longrightarrow (\mathbf{A}+\mathbf{ICI})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{I}(\mathbf{C}^{-1}+\mathbf{IA}^{-1}\mathbf{I})^{-1}\mathbf{IA}^{-1} \tag{4.92}$$

$$\Longrightarrow (\mathbf{A}+\mathbf{C})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}(\mathbf{C}^{-1}+\mathbf{A}^{-1})^{-1}\mathbf{A}^{-1} \tag{4.93}$$

4.5.3 Proof of the Sum of Two Matrices Case

Equation 4.93 can be proved as follows:

$$(\mathbf{A}+\mathbf{C})^{-1}(\mathbf{A}+\mathbf{C}) = [\mathbf{A}^{-1} - \mathbf{A}^{-1}(\mathbf{C}^{-1}+\mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}](\mathbf{A}+\mathbf{C}) \tag{4.94}$$

$$\Longrightarrow \mathbf{I} = \mathbf{A}^{-1}(\mathbf{A}+\mathbf{C}) - \mathbf{A}^{-1}(\mathbf{C}^{-1}+\mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A}+\mathbf{C}) \tag{4.95}$$

$$= \mathbf{A}^{-1}\mathbf{A} + \mathbf{A}^{-1}\mathbf{C} - \mathbf{A}^{-1}(\mathbf{C}^{-1}+\mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A}+\mathbf{C})$$

$$= \mathbf{I} + \mathbf{A}^{-1}\mathbf{C} - \mathbf{A}^{-1}(\mathbf{C}^{-1}+\mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A}+\mathbf{C})$$

$$\Longrightarrow \mathbf{I} - \mathbf{I} + \mathbf{A}^{-1}(\mathbf{C}^{-1} + \mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = \mathbf{A}^{-1}\mathbf{C} \tag{4.96}$$

$$\Longrightarrow \mathbf{A}^{-1}(\mathbf{C}^{-1} + \mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = \mathbf{A}^{-1}\mathbf{C} \tag{4.97}$$

$$\Longrightarrow \mathbf{A}\mathbf{A}^{-1}(\mathbf{C}^{-1} + \mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = \mathbf{A}\mathbf{A}^{-1}\mathbf{C} \tag{4.98}$$

$$\Longrightarrow \mathbf{I}(\mathbf{C}^{-1} + \mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = \mathbf{I}\mathbf{C} \tag{4.99}$$

$$\Longrightarrow (\mathbf{C}^{-1} + \mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = \mathbf{C} \tag{4.100}$$

$$\Longrightarrow (\mathbf{C}^{-1} + \mathbf{A}^{-1})(\mathbf{C}^{-1} + \mathbf{A}^{-1})^{-1}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = (\mathbf{C}^{-1} + \mathbf{A}^{-1})\mathbf{C} \tag{4.101}$$

$$\Longrightarrow \mathbf{I}\mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = (\mathbf{C}^{-1} + \mathbf{A}^{-1})\mathbf{C} \tag{4.102}$$

$$\Longrightarrow \mathbf{A}^{-1}(\mathbf{A} + \mathbf{C}) = (\mathbf{C}^{-1} + \mathbf{A}^{-1})\mathbf{C} \tag{4.103}$$

$$\Longrightarrow \mathbf{A}^{-1}\mathbf{A} + \mathbf{A}^{-1}\mathbf{C} = (\mathbf{C}^{-1} + \mathbf{A}^{-1})\mathbf{C} \tag{4.104}$$

$$\Longrightarrow \mathbf{I} + \mathbf{A}^{-1}\mathbf{C} = (\mathbf{C}^{-1} + \mathbf{A}^{-1})\mathbf{C} \tag{4.105}$$

$$\Longrightarrow \mathbf{I} + \mathbf{A}^{-1}\mathbf{C} = \mathbf{C}^{-1}\mathbf{C} + \mathbf{A}^{-1}\mathbf{C} \tag{4.106}$$

$$\Longrightarrow \mathbf{I} + \mathbf{A}^{-1}\mathbf{C} = \mathbf{I} + \mathbf{A}^{-1}\mathbf{C} \tag{4.107}$$

$$\Longrightarrow \mathbf{0} = \mathbf{0} \tag{4.108}$$

4.5.4 Application of the Matrix Inversion Lemma to the FDPF Constant Matrices

A power flow using the Fast Decoupled Power Flow computes and then inverts two matrices, $\left[\mathbf{B}'\right]$ and $\left[\mathbf{B}''\right]$. A contingency analysis by exhaustive enumeration performs a power flow calculation for each contingency, which by implication would involve computing and inverting $\left[\mathbf{B}'\right]$ and $\left[\mathbf{B}''\right]$ for each contingency. However, $\left[\mathbf{B}'\right]$ and $\left[\mathbf{B}''\right]$ can each be computed for the base case of all lines in, whereafter the $\left[\mathbf{B}'\right]$ and $\left[\mathbf{B}''\right]$

matrices for each contingency can be computed as the base versions plus corresponding correction matrices. The sum of a base case matrix plus correction matrix can be inverted using the special case of Matrix Inversion Lemma developed in section 4.5.2 instead of performing a full matrix inversion for each contingency [70].

Let $\left[\ \mathbf{B_0}\ \right]$ be either the $N_{buses} \times N_{buses}$ invertible matrix $\left[\ \mathbf{B}'\ \right]$ or the $N_{buses} \times N_{buses}$ invertible matrix $\left[\ \mathbf{B}''\ \right]$. Further, let $\left[\ \mathbf{B^{(n)}_{adjust}}\ \right]$ be the correction matrix that must be added to $\left[\ \mathbf{B_0}\ \right]$ to produce the outage of line n from bus ℓ to bus m, so that for any n

$$\left[\ \mathbf{B^{(n)}}\ \right] = \left[\ \mathbf{B_0}\ \right] + \left[\ \mathbf{B^{(n)}_{adjust}}\ \right] \tag{4.109}$$

Then

$$\left[\ \mathbf{B^{(n)}_{adjust}}\ \right] = b_{\ell m} \left[\ \mathbf{m^T m}\ \right] \tag{4.110}$$

Where

$$b_{\ell m} = \left[\ \mathbf{B_0}\ \right] [\ell, m] \tag{4.111}$$

The row vector $\mathbf{m}$ has two non-zero entries; $\mathbf{m}[\ell] = a$,where $a = 1$ for a transmission line and $a =$ the transformer turns ratio for a transformer, and $\mathbf{m}[\mathbf{m}] = -1$. $\left[\ \mathbf{m^T m}\ \right]$ forms the $N_{buses} \times N_{buses}$ connection matrix for line $\mathbf{n}$.

Then, using equation 4.95, and dropping matrix notation for the sake of brevity:

$$\begin{aligned}
\left(\mathbf{B_0} + \mathbf{B^{(n)}_{adjust}}\right)^{-1} &= \left(\mathbf{B_0} + b_{\ell m}\mathbf{m^T m}\right)^{-1} \\
&= \mathbf{B_0}^{-1} - \mathbf{B_0}^{-1}((b_{\ell m}\mathbf{m^T m})^{-1} + \mathbf{B_0}^{-1})^{-1}\mathbf{B_0}^{-1} \\
&= \mathbf{B_0}^{-1} - \mathbf{B_0}^{-1}(\mathbf{m}^{-1}(\mathbf{m^T})^{-1}(1/b_{\ell m}) + \mathbf{B_0}^{-1})^{-1}\mathbf{B_0}^{-1} \\
&= \mathbf{B_0}^{-1} - \mathbf{B_0}^{-1}((1/b_{\ell m})\mathbf{m}^{-1}(\mathbf{m^T})^{-1} + \mathbf{B_0}^{-1})^{-1}\mathbf{B_0}^{-1} \\
&= \mathbf{B_0}^{-1} - \mathbf{B_0}^{-1}\mathbf{m^T}((1/b_{\ell m})\mathbf{m}^{-1}(\mathbf{m^T})^{-1}\mathbf{m^T} + \mathbf{B_0}^{-1}\mathbf{m^T})^{-1}\mathbf{B_0}^{-1} \\
&= \mathbf{B_0}^{-1} - \mathbf{B_0}^{-1}\mathbf{m^T}((1/b_{\ell m})\mathbf{m}\mathbf{m}^{-1} + \mathbf{c}\mathbf{B_0}^{-1}\mathbf{m^T})^{-1}\mathbf{c}\mathbf{B_0}^{-1} \\
&= \mathbf{B_0}^{-1} - \mathbf{B_0}^{-1}\mathbf{m^T}((1/b_{\ell m}) + \mathbf{m}\mathbf{B_0}^{-1}\mathbf{m^T})^{-1}\mathbf{m}\mathbf{B_0}^{-1} \qquad (4.112)
\end{aligned}$$

Since $((1/b_{\ell m})+\mathbf{m}\mathbf{B_0^{-1}}\mathbf{m^T})^{-1}$ is a scalar, the evaluation of $(\mathbf{B_0} + \mathbf{B^{(n)}_{adjust}})^{-1}$ becomes a series of matrix-vector multiplications, scalar divisions, and matrix additions instead of having to perform another matrix inversion.

Stott and Alsac [70] define $\mathbf{x} = \mathbf{B_0}^{-1}\mathbf{m^T}$ and $c = ((1/b_{\ell m}) + \mathbf{m}\mathbf{B_0}^{-1}\mathbf{m^T})^{-1}$, giving

$$(\mathbf{B_0} + \mathbf{B^{(n)}_{adjust}})^{\mathbf{-1}} = \mathbf{B_0}^{-1} - \mathbf{x}c\mathbf{m}\mathbf{B_0}^{-1} \qquad (4.113)$$

a form similar to that derived above, which can also be written in matrix form:

$$\left(\left[\ \mathbf{B_0}\ \right] + \left[\ \mathbf{B^{(n)}_{adjust}}\ \right]\right)^{-1} = \left[\ \mathbf{B_0}\ \right]^{-1} - c\left[\ \mathbf{x}\ \right]\left[\ \mathbf{m}\ \right]\left[\ \mathbf{B_0}\ \right]^{-1} \qquad (4.114)$$

4.5.5 Substituting the Matrix Inversion Lemma into the $\Delta\theta$ and ΔV Equations

To expand Equation 4.114 into versions for $\left[\ \mathbf{B'^{(n)}}\ \right]^{-1}$ and $\left[\ \mathbf{B''^{(n)}}\ \right]^{-1}$

$$\left[\mathbf{B}'^{(\mathbf{n})} \right]^{-1} = \left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1} - c_{B'}^{(n)} \left[\mathbf{x}_{\mathbf{B}'}^{(\mathbf{n})} \right] \left[\mathbf{m}^{(\mathbf{n})} \right] \left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1} \tag{4.115}$$

$$\left[\mathbf{B}''^{(\mathbf{n})} \right]^{-1} = \left[\mathbf{B}''^{(\mathbf{0})} \right]^{-1} - c_{B''}^{(n)} \left[\mathbf{x}_{\mathbf{B}''}^{(\mathbf{n})} \right] \left[\mathbf{m}^{(\mathbf{n})} \right] \left[\mathbf{B}''^{(\mathbf{0})} \right]^{-1} \tag{4.116}$$

Note that while $c_{B'}^{(n)}$ and $c_{B''}^{(n)}$ are distinct, and $\left[\mathbf{x}_{\mathbf{B}'}^{(\mathbf{n})} \right]$ and $\left[\mathbf{x}_{\mathbf{B}''}^{(\mathbf{n})} \right]$ are distinct, the $\left[\mathbf{m}^{(\mathbf{n})} \right]$ row vector is the same in both of the preceding equations.

For the base case of Equation 4.24:

$$\left[\frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}} \right] = \left[\mathbf{B}''^{(\mathbf{0})} \right] \cdot \left[\mathbf{\Delta V^{(0)}} \right] \tag{4.117}$$

$$\left[\mathbf{\Delta V^{(0)}} \right] = \left[\mathbf{B}''^{(\mathbf{0})} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}} \right] \tag{4.118}$$

For a contingency case of Equation 4.24:

$$\left[\frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}} \right] = \left[\mathbf{B}''^{(\mathbf{n})} \right] \cdot \left[\mathbf{\Delta V^{(n)}} \right] \tag{4.119}$$

$$\left[\mathbf{\Delta V^{(n)}} \right] = \left[\mathbf{B}''^{(\mathbf{n})} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}} \right] \tag{4.120}$$

Substituting Equation 4.114 into Equation 4.120:

$$\left[\ \mathbf{\Delta V^{(n)}}\ \right] = \left(\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} - c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}\right) \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \tag{4.121}$$

After distributing terms:

$$\left[\ \mathbf{\Delta V^{(n)}}\ \right] = \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] - c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \tag{4.122}$$

If we define

$$\left[\ \mathbf{\Delta V_{temp}^{(n)}}\ \right] = \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \tag{4.123}$$

$$\begin{aligned}\left[\ \mathbf{\Delta V_{adjust}^{(n)}}\ \right] &= c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \\ &= c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{\Delta V_{temp}^{(n)}}\ \right] \\ &= c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left(\left[\ \mathbf{\Delta V_{temp}^{(n)}}\ \right][\ell] - \left[\ \mathbf{\Delta V_{temp}^{(n)}}\ \right][m]\right)\end{aligned} \tag{4.124}$$

then

$$\left[\mathbf{\Delta V^{(n)}} \right] = \left[\mathbf{\Delta V^{(n)}_{temp}} \right] - \left[\mathbf{\Delta V^{(n)}_{adjust}} \right] \tag{4.125}$$

or more concisely:

$$\begin{aligned} \left[\mathbf{\Delta V^{(n)}_{temp}} \right] &= \left[\mathbf{B''^{(0)}} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}} \right] \\ \left[\mathbf{\Delta V^{(n)}} \right] &= \left[\mathbf{\Delta V^{(n)}_{temp}} \right] - c^{(n)}_{B''} \left[\mathbf{x^{(n)}_{B''}} \right] \\ &\quad \cdot \left(\left[\mathbf{\Delta V^{(n)}_{temp}} \right] [\ell] - \left[\mathbf{\Delta V^{(n)}_{temp}} \right] [m] \right) \end{aligned} \tag{4.126}$$

Similarly,

$$\begin{aligned} \left[\mathbf{\Delta} \theta^{(\mathbf{n})}_{\mathbf{temp}} \right] &= \left[\mathbf{B'^{(0)}} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta P^{(n)}}}{\mathbf{V^{(n)}}} \right] \\ \left[\mathbf{\Delta} \theta^{(\mathbf{n})} \right] &= \left[\mathbf{\Delta} \theta^{(\mathbf{n})}_{\mathbf{temp}} \right] - c^{(n)}_{B'} \left[\mathbf{x^{(n)}_{B'}} \right] \\ &\quad \cdot \left(\left[\mathbf{\Delta} \theta^{(\mathbf{n})}_{\mathbf{temp}} \right] [\ell] - \left[\mathbf{\Delta} \theta^{(\mathbf{n})}_{\mathbf{temp}} \right] [m] \right) \end{aligned} \tag{4.127}$$

4.5.6 Stott and Alsac Approximation

Stott and Alsac define an approximation of the form [70]:

$$\left[\ \mathbf{E_1}\ \right] = \left[\ \mathbf{E_0}\ \right] - c\left[\ \mathbf{X}\ \right]\left[\ \mathbf{M}\ \right]\left[\ \mathbf{E_0}\ \right] \tag{4.128}$$

The derivation of this approximation begins with defining

$$\left[\ \mathbf{R}\ \right] = \left[\ \mathbf{B_0}\ \right]\left[\ \mathbf{E_0}\ \right] \tag{4.129}$$

to represent either

$$\left[\ \frac{\mathbf{\Delta P}}{\mathbf{V}}\ \right] = \left[\ \mathbf{B'}\ \right] \cdot \left[\ \mathbf{\Delta}\theta\ \right] \tag{4.130}$$

or

$$\left[\ \frac{\mathbf{\Delta Q}}{\mathbf{V}}\ \right] = \left[\ \mathbf{B''}\ \right] \cdot \left[\ \mathbf{\Delta V}\ \right] \tag{4.131}$$

and for which

$$\left[\ \mathbf{E_0}\ \right] = \left[\ \mathbf{B_0}\ \right]^{-1}\left[\ \mathbf{R}\ \right] \tag{4.132}$$

is a solution to Equation 4.129. They then combine the previous equation with the following two equations:

$$\left[\mathbf{B_1}\right]^{-1} = \left[\mathbf{B_0}\right]^{-1} - c\left[\mathbf{X}\right]\left[\mathbf{M}\right]\left[\mathbf{B_0}\right]^{-1} \tag{4.133}$$

$$\left[\mathbf{E_1}\right] = \left[\mathbf{B_1}\right]^{-1}\left[\mathbf{R}\right] \tag{4.134}$$

to obtain Equation 4.128. The derivation is simple enough:

$$\begin{aligned}\left[\mathbf{E_1}\right] &= \left[\mathbf{B_1}\right]^{-1}\left[\mathbf{R}\right] \\ &= \left(\left[\mathbf{B_0}\right]^{-1} - c\left[\mathbf{X}\right]\left[\mathbf{M}\right]\left[\mathbf{B_0}\right]^{-1}\right)\left[\mathbf{R}\right] \\ &= \left[\mathbf{B_0}\right]^{-1}\left[\mathbf{R}\right] - c\left[\mathbf{X}\right]\left[\mathbf{M}\right]\left[\mathbf{B_0}\right]^{-1}\left[\mathbf{R}\right]\end{aligned} \tag{4.135}$$

Substituting in Equation 4.132:

$$\left[\mathbf{E_1}\right] = \left[\mathbf{E_0}\right] - c\left[\mathbf{X}\right]\left[\mathbf{M}\right]\left[\mathbf{E_0}\right] \tag{4.136}$$

However, $\left[\mathbf{R}\right]$ in this case is either $\left[\dfrac{\mathbf{\Delta P}}{\mathbf{V}}\right]$ or $\left[\dfrac{\mathbf{\Delta Q}}{\mathbf{V}}\right]$ and

$$\begin{bmatrix} \frac{\mathbf{\Delta P_0}}{\mathbf{V_0}} \end{bmatrix} \neq \begin{bmatrix} \frac{\mathbf{\Delta P_1}}{\mathbf{V_1}} \end{bmatrix} \qquad (4.137)$$
$$\begin{bmatrix} \frac{\mathbf{\Delta Q_0}}{\mathbf{V_0}} \end{bmatrix} \neq \begin{bmatrix} \frac{\mathbf{\Delta Q_1}}{\mathbf{V_1}} \end{bmatrix}$$

Deriving the Stott and Alsac approximation, starting from the base case of Equation 4.24:

$$\begin{bmatrix} \frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}} \end{bmatrix} = \begin{bmatrix} \mathbf{B''^{(0)}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{\Delta V^{(0)}} \end{bmatrix} \qquad (4.138)$$

$$\begin{bmatrix} \mathbf{\Delta V^{(0)}} \end{bmatrix} = \begin{bmatrix} \mathbf{B''^{(0)}} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}} \end{bmatrix} \qquad (4.139)$$

For a contingency case of Equation 4.24:

$$\begin{bmatrix} \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}} \end{bmatrix} = \begin{bmatrix} \mathbf{B''^{(n)}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{\Delta V^{(n)}} \end{bmatrix} \qquad (4.140)$$

$$\begin{bmatrix} \mathbf{\Delta V^{(n)}} \end{bmatrix} = \begin{bmatrix} \mathbf{B''^{(n)}} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}} \end{bmatrix} \qquad (4.141)$$

Substituting Equation 4.114 into Equation 4.120:

$$\begin{aligned}\left[\ \mathbf{\Delta V^{(n)}}\ \right] &= \left(\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} - c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}\right) \\ &\cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \\ &= \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \\ &- c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right]\end{aligned} \tag{4.142}$$

Subtracting Equation 4.139 from Equation 4.142:

$$\begin{aligned}\left[\ \mathbf{\Delta V^{(n)}}\ \right] - \left[\ \mathbf{\Delta V^{(0)}}\ \right] &= \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \\ &- \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}}\ \right] \\ &- c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right]\end{aligned} \tag{4.143}$$

If we neglect the difference between $\left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right]$ and $\left[\ \frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}}\ \right]$, then

$$\begin{aligned}\left[\ \mathbf{\Delta V^{(n)}}\ \right] - \left[\ \mathbf{\Delta V^{(0)}}\ \right] &= -\, c_{B''}^{(n)}\left[\ \mathbf{x_{B''}^{(n)}}\ \right]\left[\ \mathbf{m^{(n)}}\ \right]\left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \\ &\cdot \left[\ \frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}}\ \right]\end{aligned} \tag{4.144}$$

Substituting in Equation 4.139:

$$\left[\ \mathbf{\Delta V^{(n)}}\ \right] - \left[\ \mathbf{\Delta V^{(0)}}\ \right] = -\, c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \left[\ \mathbf{m^{(n)}}\ \right] \cdot \left[\ \mathbf{\Delta V^{(0)}}\ \right] \tag{4.145}$$

Thus the contingency case of Equation 4.24 follows the form of the Stott and Alsac approximation [70]:

$$\left[\ \mathbf{\Delta V^{(n)}}\ \right] = \left[\ \mathbf{\Delta V^{(0)}}\ \right] - c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \left[\ \mathbf{m^{(n)}}\ \right] \cdot \left[\ \mathbf{\Delta V^{(0)}}\ \right] \tag{4.146}$$

or

$$\begin{aligned} \left[\ \mathbf{\Delta V^{(n)}}\ \right] = \left[\ \mathbf{\Delta V^{(0)}}\ \right] - c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \\ \cdot \left(\left[\ \mathbf{\Delta V^{(0)}}\ \right] [\ell] - \left[\ \mathbf{\Delta V^{(0)}}\ \right] [m] \right) \end{aligned} \tag{4.147}$$

Similarly, the contingency case of Equation 4.15, neglecting the difference between $\left[\ \dfrac{\mathbf{\Delta P^{(n)}}}{\mathbf{V^{(n)}}}\ \right]$ and $\left[\ \dfrac{\mathbf{\Delta P^{(0)}}}{\mathbf{V^{(0)}}}\ \right]$ becomes [70]:

$$\left[\ \mathbf{\Delta}\theta^{\mathbf{(n)}}\ \right] = \left[\ \mathbf{\Delta}\theta^{\mathbf{(0)}}\ \right] - c_{B'}^{(n)} \left[\ \mathbf{x_{B'}^{(n)}}\ \right] \left[\ \mathbf{m^{(n)}}\ \right] \cdot \left[\ \mathbf{\Delta}\theta^{\mathbf{(0)}}\ \right] \tag{4.148}$$

or

$$\left[\; \boldsymbol{\Delta}\theta^{(\mathbf{n})} \; \right] = \left[\; \boldsymbol{\Delta}\theta^{(\mathbf{0})} \; \right] - c_{B'}^{(n)} \left[\; \mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \; \right] \cdot \left(\left[\; \boldsymbol{\Delta}\theta^{(\mathbf{0})} \; \right] [\ell] - \left[\; \boldsymbol{\Delta}\theta^{(\mathbf{0})} \; \right] [m] \right) \tag{4.149}$$

Difficulties with the Stott and Alsac Approximation

The approximation described above has very useful properties in that the (N-1) contingency solutions can be computed without requiring $\left[\; \mathbf{B}'^{(\mathbf{n})} \; \right]^{-1}$ and $\left[\; \mathbf{B}''^{(\mathbf{n})} \; \right]^{-1}$ or even $\left[\; \mathbf{B}'^{(\mathbf{0})} \; \right]^{-1}$ and $\left[\; \mathbf{B}''^{(\mathbf{0})} \; \right]^{-1}$ within iterations of the contingency solution. $\left[\; \mathbf{B}'^{(\mathbf{0})} \; \right]^{-1}$ and $\left[\; \mathbf{B}''^{(\mathbf{0})} \; \right]^{-1}$ only are needed for this method at all, and only then to pre-compute scalar constants and vector constants for each contingency.

However, as described above the approximation depends on neglecting the difference between $\left[\; \frac{\boldsymbol{\Delta}\mathbf{Q}^{(\mathbf{n})}}{\mathbf{V}^{(\mathbf{n})}} \; \right]$ and $\left[\; \frac{\boldsymbol{\Delta}\mathbf{Q}^{(\mathbf{0})}}{\mathbf{V}^{(\mathbf{0})}} \; \right]$ in computing the ΔV equation, as shown in Equation 4.144, and on neglecting the difference between $\left[\; \frac{\boldsymbol{\Delta}\mathbf{P}^{(\mathbf{n})}}{\mathbf{V}^{(\mathbf{n})}} \; \right]$ and $\left[\; \frac{\boldsymbol{\Delta}\mathbf{P}^{(\mathbf{0})}}{\mathbf{V}^{(\mathbf{0})}} \; \right]$ in computing the $\Delta\theta$ equation. This may significantly affect the number of iterations required for convergence.

Examination of Equations 4.128, 4.146 and 4.148 shows that under this approximation, the equations for $\left[\; \boldsymbol{\Delta}\mathbf{V}^{(\mathbf{n})} \; \right]$ and $\left[\; \boldsymbol{\Delta}\theta^{(\mathbf{n})} \; \right]$ do not depend on data from the current iteration of the solution method at all, but only on the pre-computed constants and base case data. This means $\left[\; \boldsymbol{\Delta}\mathbf{V}^{(\mathbf{n})} \; \right]$ and $\left[\; \boldsymbol{\Delta}\theta^{(\mathbf{n})} \; \right]$ can be pre-computed, and their values are entirely decoupled from the iterations of the solver. It bears consideration whether it will remain stable under all conditions.

Further, while Stott and Alsac state this approximation can be applied the set of (N-2) contingencies using the the base case (N-0) data by updating the method for multiple outages, they originally found it faster than inverting a new set of $\left[\mathbf{B}'^{(\mathbf{n})} \right]^{-1}$, and $\left[\mathbf{B}''^{(\mathbf{n})} \right]^{-1}$ matrices for at most three multiple outages [70], presumably because of additional iterations required for convergence under the approximate method.

The speed of the approximate method makes it of interest for the work developed in this thesis. However, the equations for $\Delta\theta$ and ΔV developed in Section 4.5.5 make a better basis for (N-2) and (N-x) contingency cases, and may simply prove more practical as highly-optimized GPGPGU sparse matrix routines become widely available. Methods for (N-x) will be discussed further in Chapter 5.

An alternative pair of equations for $\Delta\theta$ and ΔV that are not completely decoupled from the iterations of the contingency solver could be formed as follows:

$$\begin{aligned} \left[\mathbf{\Delta}\theta^{(\mathbf{n})} \right] &= \left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta P}^{(\mathbf{0})}}{\mathbf{V}^{(\mathbf{0})}} \right] \\ &- c_{B'}^{(n)} \left[\mathbf{x}_{\mathbf{B}'}^{(\mathbf{n})} \right] \left[\mathbf{m}^{(\mathbf{n})} \right] \left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta P}^{(\mathbf{n})}}{\mathbf{V}^{(\mathbf{n})}} \right] \\ &= \left[\mathbf{\Delta}\theta^{(\mathbf{0})} \right] - c_{B'}^{(n)} \left[\mathbf{x}_{\mathbf{B}'}^{(\mathbf{n})} \right] \left[\mathbf{m}^{(\mathbf{n})} \right] \left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1} \cdot \left[\frac{\mathbf{\Delta P}^{(\mathbf{n})}}{\mathbf{V}^{(\mathbf{n})}} \right] \end{aligned} \tag{4.150}$$

$$\left[\begin{array}{c}\mathbf{\Delta V^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]^{-1}\cdot\left[\begin{array}{c}\frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}}\end{array}\right] - c_{B''}^{(n)}\left[\begin{array}{c}\mathbf{x_{B''}^{(n)}}\end{array}\right]\left[\begin{array}{c}\mathbf{m^{(n)}}\end{array}\right]\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]^{-1}\cdot\left[\begin{array}{c}\frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\end{array}\right]$$
$$= \left[\begin{array}{c}\mathbf{\Delta V^{(0)}}\end{array}\right] - c_{B''}^{(n)}\left[\begin{array}{c}\mathbf{x_{B''}^{(n)}}\end{array}\right]\left[\begin{array}{c}\mathbf{m^{(n)}}\end{array}\right]\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]^{-1}\cdot\left[\begin{array}{c}\frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\end{array}\right] \quad (4.151)$$

These are not of interest for the work developed in this thesis. Since the use of these equations would require $\left[\begin{array}{c}\mathbf{B'^{(0)}}\end{array}\right]^{-1}$ and $\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]^{-1}$ within iterations of the contingency solution, it is preferable to use the more accurate equations for $\Delta\theta$ and ΔV developed in Section 4.5.5.

4.6 FPDF Contingency Algorithm with Stott and Alsac Approximation

This section will walk through computing a single contingency n in which the line from bus ℓ to bus m is out of service, using Equations 4.148 and 4.146 developed in Section 4.5.6 for the $\Delta\theta$ and ΔV calculations respectively. Section 4.3 gives the solution for the base case of the system, with no lines out of service. For this section, it is presumed that the base case for no lines out of services has been solved and the data available, with the tolerances for convergence of the base case set very small.

In this version of the algorithm, the $\left[\begin{array}{c}\mathbf{G_{bus}}\end{array}\right]$ and $\left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right]$ matrices are the only sparse matrices.

4.6.1 Inputs to the FPDF Contingency Algorithm with Stott and Alsac Approximation

- $\left[\begin{array}{c}\mathbf{V_0^{(n)}}\end{array}\right]$ is an $N_{buses} \times 1$ vector of scalar voltage magnitudes at each bus from the solution of the base case
- $\left[\begin{array}{c}\theta_{\mathbf{0}}^{\mathbf{(n)}}\end{array}\right]$ is an $N_{buses} \times 1$ vector of scalar voltage angles at each bus from the solution of the base case
- $\left[\begin{array}{c}\mathbf{G_{bus}}\end{array}\right]$ is an $N_{buses} \times N_{buses}$ matrix of the real parts of the entries in the $\left[\begin{array}{c}\mathbf{Y_{bus}^{\rightarrow}}\end{array}\right]$ matrix for the base case
- $\left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right]$ is an $N_{buses} \times N_{buses}$ matrix of the imaginary parts of the entries in the $\left[\begin{array}{c}\mathbf{Y_{bus}^{\rightarrow}}\end{array}\right]$ matrix for the base case
- $\left[\begin{array}{c}\mathbf{P_{sched}}\end{array}\right]$ is an $N_{buses} \times 1$ vector of scalar scheduled active power at each bus
- $\left[\begin{array}{c}\mathbf{Q_{sched}}\end{array}\right]$ is an $N_{buses} \times 1$ vector of scalar scheduled reactive power at each bus
- $c_{B'}^{(n)}$ and $c_{B''}^{(n)}$ are pre-computed constants for the $\mathbf{\Delta}\theta$ and $\mathbf{\Delta V}$ update equations
- $\left[\begin{array}{c}\mathbf{x_{B'}^{(n)}}\end{array}\right]$ and $\left[\begin{array}{c}\mathbf{x_{B''}^{(n)}}\end{array}\right]$ pre-computed $N_{buses} \times 1$ vectors for the $\mathbf{\Delta}\theta$ and $\mathbf{\Delta V}$ update equations
- $\left[\begin{array}{c}\mathbf{\Delta}\theta^{\mathbf{(0)}}\end{array}\right]$ is an $N_{buses} \times 1$ vector of scalar voltage angle deltas at each bus from the solution of the base case
- $\left[\begin{array}{c}\mathbf{\Delta V^{(0)}}\end{array}\right]$ is an $N_{buses} \times 1$ vector of scalar voltage magnitude deltas at each bus from the solution of the base case

4.6.2 The $P-\theta$ Iteration:

The $P-\theta$ iteration:

$$\left[\ \mathbf{V_0'^{(n)}}\ \right] = \left[\ \mathbf{V_0^{(n)}}\ \right] *.\left(\cos.\left[\ \theta_{\mathbf{0}}^{\mathbf{(n)}}\ \right]\right) \tag{4.152}$$

$$\left[\ \mathbf{V_0''^{(n)}}\ \right] = \left[\ \mathbf{V_0^{(n)}}\ \right] *.\left(\sin.\left[\ \theta_{\mathbf{0}}^{\mathbf{(n)}}\ \right]\right) \tag{4.153}$$

$$\left[\ \mathbf{I_{1/2}'^{(n)}}\ \right] = \left[\ \mathbf{G_{bus}}\ \right]\cdot\left[\ \mathbf{V_0'^{(n)}}\ \right] - \left[\ \mathbf{B_{bus}}\ \right]\cdot\left[\ \mathbf{V_0''^{(n)}}\ \right] \tag{4.154}$$

$$\left[\ \mathbf{I_{1/2}''^{(n)}}\ \right] = \left[\ \mathbf{G_{bus}}\ \right]\cdot\left[\ \mathbf{V_0''^{(n)}}\ \right] + \left[\ \mathbf{B_{bus}}\ \right]\cdot\left[\ \mathbf{V_0'^{(n)}}\ \right] \tag{4.155}$$

$$\begin{aligned}\mathbf{I_{adjust,1/2}'^{(n)}} = & \left[\ \mathbf{G_{bus}}\ \right][\ell,m]\cdot\left(\left[\ \mathbf{V_0'^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_0'^{(n)}}\ \right][m]\right) \\ & - \left[\ \mathbf{B_{bus}}\ \right][\ell,m]\cdot\left(\left[\ \mathbf{V_0''^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_0''^{(n)}}\ \right][m]\right)\end{aligned} \tag{4.156}$$

$$\begin{aligned}\left[\ \mathbf{I_{1/2}'^{(n)}}\ \right][\ell] &= \left[\ \mathbf{I_{1/2}'^{(n)}}\ \right][\ell] + \mathbf{I_{adjust,1/2}'^{(n)}} \\ \left[\ \mathbf{I_{1/2}'^{(n)}}\ \right][m] &= \left[\ \mathbf{I_{1/2}'^{(n)}}\ \right][m] - \mathbf{I_{adjust,1/2}'^{(n)}}\end{aligned} \tag{4.157}$$

$$\begin{aligned}\mathbf{I_{adjust,1/2}''^{(n)}} = & \left[\ \mathbf{G_{bus}}\ \right][\ell,m]\cdot\left(\left[\ \mathbf{V_0''^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_0''^{(n)}}\ \right][m]\right) \\ & + \left[\ \mathbf{B_{bus}}\ \right][\ell,m]\cdot\left(\left[\ \mathbf{V_0'^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_0'^{(n)}}\ \right][m]\right)\end{aligned} \tag{4.158}$$

$$\begin{aligned}\left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] &= \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] + \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust,1/2}} \\ \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m] &= \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m] - \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust,1/2}}\end{aligned} \tag{4.159}$$

$$\left[\ \mathbf{P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}}\ \right] *\cdot \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] + \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}}\ \right] *\cdot \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] \tag{4.160}$$

(4.161)

$$\left[\ \mathbf{\Delta P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{P}_{\mathbf{sched}}\ \right] - \left[\ \mathbf{P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] \tag{4.162}$$

$$\Delta P^{(n)}_{max,1} = \max\left(\left[\ \mathbf{\Delta P}^{(\mathbf{n})}_{\mathbf{1}}\ \right]\right) \tag{4.163}$$

$$\begin{aligned}\left[\ \mathbf{\Delta}\theta^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{\Delta}\theta^{(\mathbf{0})}\ \right] - c^{(n)}_{B'}\left[\ \mathbf{x}^{(\mathbf{n})}_{\mathbf{B'}}\ \right] \\ \cdot\ \left(\left[\ \mathbf{\Delta}\theta^{(\mathbf{0})}\ \right][\ell] - \left[\ \mathbf{\Delta}\theta^{(\mathbf{0})}\ \right][m]\right)\end{aligned} \tag{4.164}$$

$$\left[\ \theta^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \theta^{(\mathbf{n})}_{\mathbf{0}}\ \right] + \left[\ \mathbf{\Delta}\theta^{(\mathbf{n})}_{\mathbf{1}}\ \right] \tag{4.165}$$

4.6.3 The $Q-V$ Iteration:

The $Q-V$ iteration:

$$\left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] = \left[\ \mathbf{V}^{(\mathbf{n})}_{\mathbf{0}}\ \right] *. \left(\cos . \left[\ \theta^{(\mathbf{n})}_{\mathbf{1}}\ \right]\right) \tag{4.166}$$

$$\left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] = \left[\ \mathbf{V}^{(\mathbf{n})}_{\mathbf{0}}\ \right] *. \left(\sin . \left[\ \theta^{(\mathbf{n})}_{\mathbf{1}}\ \right]\right) \tag{4.167}$$

$$\left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{G}_{\mathbf{bus}}\ \right] \cdot \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] - \left[\ \mathbf{B}_{\mathbf{bus}}\ \right] \cdot \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] \tag{4.168}$$

$$\left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{G}_{\mathbf{bus}}\ \right] \cdot \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] + \left[\ \mathbf{B}_{\mathbf{bus}}\ \right] \cdot \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] \tag{4.169}$$

$$\begin{aligned} \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} = & \left[\ \mathbf{G}_{\mathbf{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) \\ & - \left[\ \mathbf{B}_{\mathbf{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) \end{aligned} \tag{4.170}$$

$$\begin{aligned} \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][\ell] &= \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][\ell] + \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} \\ \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][m] &= \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][m] - \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} \end{aligned} \tag{4.171}$$

$$\begin{aligned} \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} = & \left[\ \mathbf{G}_{\mathbf{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) \\ & + \left[\ \mathbf{B}_{\mathbf{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) \end{aligned} \tag{4.172}$$

$$\left[\begin{array}{c}\mathbf{I_1''^{(n)}}\end{array}\right][\ell] = \left[\begin{array}{c}\mathbf{I_1''^{(n)}}\end{array}\right][\ell] + \mathbf{I''^{(n)}_{adjust,1}} \tag{4.173}$$
$$\left[\begin{array}{c}\mathbf{I_1''^{(n)}}\end{array}\right][m] = \left[\begin{array}{c}\mathbf{I_1''^{(n)}}\end{array}\right][m] - \mathbf{I''^{(n)}_{adjust,1}}$$

$$\left[\begin{array}{c}\mathbf{Q_1^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{V''^{(n)}_{1/2}}\end{array}\right] *\cdot \left[\begin{array}{c}\mathbf{I_1'^{(n)}}\end{array}\right] - \left[\begin{array}{c}\mathbf{V'^{(n)}_{1/2}}\end{array}\right] *\cdot \left[\begin{array}{c}\mathbf{I_1''^{(n)}}\end{array}\right] \tag{4.174}$$

$$\left[\begin{array}{c}\mathbf{\Delta Q_1^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{Q_{sched}}\end{array}\right] - \left[\begin{array}{c}\mathbf{Q_1^{(n)}}\end{array}\right] \tag{4.175}$$

$$\Delta Q^{(n)}_{max,1} = \max\left(\left[\begin{array}{c}\mathbf{\Delta Q_1^{(n)}}\end{array}\right]\right) \tag{4.176}$$

$$\left[\begin{array}{c}\mathbf{\Delta V_1^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{\Delta V^{(0)}}\end{array}\right] - c^{(n)}_{B''}\left[\begin{array}{c}\mathbf{x^{(n)}_{B''}}\end{array}\right] \tag{4.177}$$
$$\cdot\left(\left[\begin{array}{c}\mathbf{\Delta V^{(0)}}\end{array}\right][\ell] - \left[\begin{array}{c}\mathbf{\Delta V^{(0)}}\end{array}\right][m]\right)$$

$$\left[\begin{array}{c}\mathbf{V_1^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{V_0^{(n)}}\end{array}\right] + \left[\begin{array}{c}\mathbf{\Delta V_1^{(n)}}\end{array}\right] \tag{4.178}$$

If ΔP_{max} and ΔQ_{max} have not converged to within tolerance, iterate by returning to Equation 4.152 , which now becomes:

$$\left[\ \mathbf{V_1'^{(n)}}\ \right] = \left[\ \mathbf{V_1^{(n)}}\ \right] *. \left(\cos.\left[\ \theta_{\mathbf{1}}^{(\mathbf{n})}\ \right]\right) \tag{4.179}$$

$$\left[\ \mathbf{V_1''^{(n)}}\ \right] = \left[\ \mathbf{V_1^{(n)}}\ \right] *. \left(\sin.\left[\ \theta_{\mathbf{1}}^{(\mathbf{n})}\ \right]\right) \tag{4.180}$$

4.7 FPDF Contingency Algorithm, Full $\Delta\theta$ and ΔV

This section will walk through computing a single contingency n in which the line from bus ℓ to bus m is out of service, using Equations 4.126 and 4.127 for the $\Delta\theta$ and ΔV calculations respectively. Section 4.3 gives the solution for the base case of the system, with no lines out of service. For this section, it is presumed that the base case for no lines out of services has been solved and the data available, with the tolerances for convergence of the base case set very small.

This version is more mathematically accurate than Section 4.6 in that it does not depend on neglecting the difference between $\left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right]$ and $\left[\ \frac{\mathbf{\Delta Q^{(0)}}}{\mathbf{V^{(0)}}}\ \right]$ in computing the ΔV equation, as shown in Equation 4.144, and on neglecting the difference between $\left[\ \frac{\mathbf{\Delta P^{(n)}}}{\mathbf{V^{(n)}}}\ \right]$ and $\left[\ \frac{\mathbf{\Delta P^{(0)}}}{\mathbf{V^{(0)}}}\ \right]$ in computing the $\Delta\theta$ equation. As a result, this version is mathematically more nearly equivalent to the conventional FDPF. This may significantly reduce the number of iterations required for convergence.

Further, while method in Section 4.6 can be applied the set of (N-2) contingencies using the the base case (N-0) data by updating the method for multiple outages, the approximation involved does affect convergence enough that Stott and Alsac originally found it faster than inverting a new set of $\left[\ \mathbf{B'^{(n)}}\ \right]^{-1}$, and $\left[\ \mathbf{B''^{(n)}}\ \right]^{-1}$ matrices for at most three multiple outages [70]. As a result, both versions are of use for computing (N-x) contingencies.

Methods for (N-x) will be discussed further in Chapter 5.

In this version of the algorithm the large sparse matrices that must be transported to the GPGPU are four instead of two: $\left[\ \mathbf{G_{bus}}\ \right]$, $\left[\ \mathbf{B_{bus}}\ \right]$, $\left[\ \mathbf{B'^{(n)}}\ \right]^{-1}$, and $\left[\ \mathbf{B''^{(n)}}\ \right]^{-1}$. Additionally, there are other aspects to this version that are not as amenable to some of the methods developed in Chapter 5.

4.7.1 Inputs to the FPDF Contingency Algorithm, Full $\Delta\theta$ and ΔV

- $\left[\ \mathbf{V_0^{(n)}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar voltage magnitudes at each bus from the solution of the base case
- $\left[\ \theta_{\mathbf{0}}^{(\mathbf{n})}\ \right]$ is an $N_{buses} \times 1$ vector of scalar voltage angles at each bus from the solution of the base case
- $\left[\ \mathbf{G_{bus}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of the real parts of the entries in the $\left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right]$ matrix for the base case
- $\left[\ \mathbf{B_{bus}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of the imaginary parts of the entries in the $\left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right]$ matrix for the base case
- $\left[\ \mathbf{P_{sched}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled active power at each bus
- $\left[\ \mathbf{Q_{sched}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled reactive power at each bus
- $\left[\ \mathbf{B'^{(0)}}\ \right]^{-1}$ is a pre-computed inverse of a FDPF constant matrix from the solution of the base case

- $\left[\begin{array}{c} \mathbf{B}''^{(\mathbf{0})} \end{array}\right]^{-1}$ is a pre-computed inverse of a FDPF constant matrix from the solution of the base case
- $c_{B'}^{(n)}$ and $c_{B''}^{(n)}$ are pre-computed scalar constants for the $\boldsymbol{\Delta}\theta$ and $\boldsymbol{\Delta}\mathbf{V}$ update equations
- $\left[\begin{array}{c} \mathbf{x}_{\mathbf{B}'}^{(\mathbf{n})} \end{array}\right]$ and $\left[\begin{array}{c} \mathbf{x}_{\mathbf{B}''}^{(\mathbf{n})} \end{array}\right]$ pre-computed $N_{buses} \times 1$ vectors of scalar values for the $\boldsymbol{\Delta}\theta$ and $\boldsymbol{\Delta}\mathbf{V}$ update equations

4.7.2 The $P - \theta$ Iteration:

The $P - \theta$ iteration:

$$\left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}'^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}^{(\mathbf{n})} \end{array}\right] *. \left(\cos . \left[\begin{array}{c} \theta_{\mathbf{0}}^{(\mathbf{n})} \end{array}\right]\right) \tag{4.181}$$

$$\left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}''^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}^{(\mathbf{n})} \end{array}\right] *. \left(\sin . \left[\begin{array}{c} \theta_{\mathbf{0}}^{(\mathbf{n})} \end{array}\right]\right) \tag{4.182}$$

$$\left[\begin{array}{c} \mathbf{I}_{\mathbf{1/2}}'^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}'^{(\mathbf{n})} \end{array}\right] - \left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}''^{(\mathbf{n})} \end{array}\right] \tag{4.183}$$

$$\left[\begin{array}{c} \mathbf{I}_{\mathbf{1/2}}''^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}''^{(\mathbf{n})} \end{array}\right] + \left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}'^{(\mathbf{n})} \end{array}\right] \tag{4.184}$$

$$\begin{aligned} \mathbf{I}_{\mathbf{adjust,1/2}}'^{(\mathbf{n})} = & \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right][\ell, m] \cdot \left(\left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}'^{(\mathbf{n})} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}'^{(\mathbf{n})} \end{array}\right][m]\right) \\ & - \left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right][\ell, m] \cdot \left(\left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}''^{(\mathbf{n})} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V}_{\mathbf{0}}''^{(\mathbf{n})} \end{array}\right][m]\right) \end{aligned} \tag{4.185}$$

$$\begin{aligned} \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] &= \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] + \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust,1/2}} \\ \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m] &= \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m] - \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust,1/2}} \end{aligned} \tag{4.186}$$

$$\begin{aligned} \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust,1/2}} = {} & \left[\ \mathbf{G_{bus}}\ \right][\ell,m] \cdot \left(\left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}}\ \right][\ell] - \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}}\ \right][m]\right) \\ & + \left[\ \mathbf{B_{bus}}\ \right][\ell,m] \cdot \left(\left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}}\ \right][\ell] - \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}}\ \right][m]\right) \end{aligned} \tag{4.187}$$

$$\begin{aligned} \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] &= \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] + \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust,1/2}} \\ \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m] &= \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m] - \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust,1/2}} \end{aligned} \tag{4.188}$$

$$\left[\ \mathbf{P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{0}}\ \right] *\cdot \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] + \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{0}}\ \right] *\cdot \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] \tag{4.189}$$

$$\left[\ \mathbf{\Delta P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{P_{sched}}\ \right] - \left[\ \mathbf{P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] \tag{4.190}$$

$$\Delta P^{(n)}_{max,1} = \max\left(\left[\ \mathbf{\Delta P}^{(\mathbf{n})}_{\mathbf{1}}\ \right]\right) \tag{4.191}$$

$$\left[\ \frac{\mathbf{\Delta P}^{(\mathbf{n})}_{\mathbf{1}}}{\mathbf{V}^{(\mathbf{n})}_{\mathbf{1}}}\ \right] = \left[\ \mathbf{\Delta P}^{(\mathbf{n})}_{\mathbf{1}}\ \right] /\cdot \left[\ \mathbf{V}^{(\mathbf{n})}_{\mathbf{0}}\ \right] \tag{4.192}$$

$$\left[\boldsymbol{\Delta}\theta^{(\mathbf{n})}_{\mathbf{temp,1}} \right] = \left[\mathbf{B}' \right]^{-1} \cdot \left[\frac{\boldsymbol{\Delta}\mathbf{P}^{(\mathbf{n})}_{\mathbf{1}}}{\mathbf{V}^{(\mathbf{n})}_{\mathbf{1}}} \right] \tag{4.193}$$

$$\left[\boldsymbol{\Delta}\theta^{(\mathbf{n})}_{\mathbf{1}} \right] = \left[\boldsymbol{\Delta}\theta^{(\mathbf{n})}_{\mathbf{temp,1}} \right] - c^{(n)}_{B'} \left[\mathbf{x}^{(\mathbf{n})}_{\mathbf{B'}} \right] \cdot \left(\left[\boldsymbol{\Delta}\theta^{(\mathbf{n})}_{\mathbf{temp,1}} \right] [\ell] - \left[\boldsymbol{\Delta}\theta^{(\mathbf{n})}_{\mathbf{temp,1}} \right] [m] \right)$$

$$\left[\theta^{(\mathbf{n})}_{\mathbf{1}} \right] = \left[\theta^{(\mathbf{n})}_{\mathbf{0}} \right] + \left[\boldsymbol{\Delta}\theta^{(\mathbf{n})}_{\mathbf{1}} \right] \tag{4.194}$$

4.7.3 The $Q - V$ Iteration:

The $Q - V$ iteration:

$$\left[\mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}} \right] = \left[\mathbf{V}^{(\mathbf{n})}_{\mathbf{0}} \right] * . \left(\cos . \left[\theta^{(\mathbf{n})}_{\mathbf{1}} \right] \right) \tag{4.195}$$

$$\left[\mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}} \right] = \left[\mathbf{V}^{(\mathbf{n})}_{\mathbf{0}} \right] * . \left(\sin . \left[\theta^{(\mathbf{n})}_{\mathbf{1}} \right] \right) \tag{4.196}$$

$$\left[\mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}} \right] = \left[\mathbf{G}_{\mathbf{bus}} \right] \cdot \left[\mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}} \right] - \left[\mathbf{B}_{\mathbf{bus}} \right] \cdot \left[\mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}} \right] \tag{4.197}$$

$$\left[\mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}} \right] = \left[\mathbf{G}_{\mathbf{bus}} \right] \cdot \left[\mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}} \right] + \left[\mathbf{B}_{\mathbf{bus}} \right] \cdot \left[\mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}} \right] \tag{4.198}$$

$$\mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} = \left[\ \mathbf{G_{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) - \left[\ \mathbf{B_{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) \tag{4.199}$$

$$\begin{aligned} \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][\ell] &= \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][\ell] + \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} \\ \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][m] &= \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right][m] - \mathbf{I}'^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} \end{aligned} \tag{4.200}$$

$$\mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} = \left[\ \mathbf{G_{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) + \left[\ \mathbf{B_{bus}}\ \right][\ell, m] \cdot \left(\left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][\ell] - \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right][m]\right) \tag{4.201}$$

$$\begin{aligned} \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}}\ \right][\ell] &= \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}}\ \right][\ell] + \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} \\ \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}}\ \right][m] &= \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}}\ \right][m] - \mathbf{I}''^{(\mathbf{n})}_{\mathbf{adjust},\mathbf{1}} \end{aligned} \tag{4.202}$$

$$\left[\ \mathbf{Q}^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{V}''^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] * \cdot \left[\ \mathbf{I}'^{(\mathbf{n})}_{\mathbf{1}}\ \right] - \left[\ \mathbf{V}'^{(\mathbf{n})}_{\mathbf{1/2}}\ \right] * \cdot \left[\ \mathbf{I}''^{(\mathbf{n})}_{\mathbf{1}}\ \right] \tag{4.203}$$

$$\left[\ \mathbf{\Delta Q}^{(\mathbf{n})}_{\mathbf{1}}\ \right] = \left[\ \mathbf{Q_{sched}}\ \right] - \left[\ \mathbf{Q}^{(\mathbf{n})}_{\mathbf{1}}\ \right] \tag{4.204}$$

$$\Delta Q^{(n)}_{max,1} = \max\left(\left[\ \mathbf{\Delta Q^{(n)}_1}\ \right]\right) \tag{4.205}$$

$$\left[\ \frac{\mathbf{\Delta Q^{(n)}_1}}{\mathbf{V^{(n)}_1}}\ \right] = \left[\ \mathbf{\Delta Q^{(n)}_1}\ \right] /. \left[\ \mathbf{V^{(n)}_{1/2}}\ \right] \tag{4.206}$$

$$\begin{aligned} \left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right] &= \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \\ \left[\ \mathbf{\Delta V^{(n)}}\ \right] &= \left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right] - c^{(n)}_{B''} \left[\ \mathbf{x^{(n)}_{B''}}\ \right] \\ &\cdot \left(\left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right][\ell] - \left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right][m]\right) \end{aligned} \tag{4.207}$$

$$\left[\ \mathbf{V^{(n)}_1}\ \right] = \left[\ \mathbf{V^{(n)}_0}\ \right] + \left[\ \mathbf{\Delta V^{(n)}_1}\ \right] \tag{4.208}$$

If ΔP_{max} and ΔQ_{max} have not converged to within tolerance, iterate by returning to Equation 4.181 , which now becomes:

$$\left[\ \mathbf{V'^{(n)}_1}\ \right] = \left[\ \mathbf{V^{(n)}_1}\ \right] *. \left(\cos.\left[\ \theta^{(n)}_1\ \right]\right) \tag{4.209}$$

$$\left[\ \mathbf{V''^{(n)}_1}\ \right] = \left[\ \mathbf{V^{(n)}_1}\ \right] *. \left(\sin.\left[\ \theta^{(n)}_1\ \right]\right) \tag{4.210}$$

4.8 Summary

The beginning of this chapter outlined the motivations for the design choices underlying the method developed in this thesis. While GPGPUs and their associated toolsets have developed substantially since this work was begun, many of the fundamental properties that drove the design criteria for this method remain the same. GPGPUs are still designed primarily for rudimentary arithmetic at the transistor level. Moving data to and from the GPGPU from the computer system main memory can still be a fundamental determiner of application speed. The architecture of GPGPUs remains the SIMD-within MIMD architecture described in Chapter 3. As a result, the method developed in this thesis remains of interest even with improvements to the available technology.

This chapter also outlined the basic FDPF algorithm and then showed alterations to that algorithm designed to both deal with the limits of the GPGPU and facilitate slicing across the set of (N-1) contingencies to create blocks of dense vectorized computation. One proposed method that uses a fast approximation was outlined for a single contingency in Section 4.6. A second proposed method without the approximation was outlined for a single contingency in Section 4.7 Further details and methods for computing across contingencies will be the focus of Chapter 5.

Chapter 5

Details of the Proposed Method

This chapter continues from Chapter 4 and gives additional details of the proposed method. Continuing the discussion of computing the set of (N-1) contingencies, the proposed method exploits as much as possible instances where the same data can be used simultaneously in computations for all of the set of (N-1) contingencies. The routines for the computing the algorithm in Section 4.6 are primarily of two types:

- Routines in which the same data is needed across all (N-1) contingencies.
- Routines in which the data to be worked upon is unique to each contingency.

Further, while most computations take place across repeated iterations of the FDPF contingency algorithm proposed here, there are some that can be pre-computed before iterations of the solver begin.

Nearly all the routines discussed in this chapter involve manipulation of large blocks of dense vectors. Depending on the GPGPU used and the size of the power system being studied, all the data for a given routine may fit on the GPGPU at one time, or it may not. To aid in clarity, the routines below will not, in the main, include tiling strategies for breaking the data into sections that will fit in GPGPU memory, though comments on which data to prioritize may be included. Since the functionality to interleave GPGPU computation

with transporting data between the computer system main memory and the GPGPU is now fairly standard (most GPGPUs sold after 2010 have this capability), it is recommended that this functionality be employed as much as possible, both in tiling data into sections for individual routines and in preparing data needed by the next routine.

This chapter first details some common routines that are used in multiple instances in the methodology proposed. The next section discusses pre-computation of data that can be prepared outside of the main loop of the iterative FDPF solution algorithm. The next section is a walk-through of the proposed method for computing the set of (N-1) contingencies, detailing methods for slicing across contingencies to create large blocks of dense vector operations. The chapter concludes with a discussion of applying the method developed here to (N-x) contingency analysis and a chapter summary.

5.1 Common Routines

Certain routines that are used repeatedly throughout the method are discussed in this section.

5.1.1 A Sparse Matrix Multiplied by a Block of Dense Vectors

Let there be a $N_{buses} \times N_{buses}$ matrix $\left[\mathbf{S} \right]$, which is to be multiplied by a different $N_{buses} \times 1$ vector $\left[\mathbf{d^{(n)}} \right]$ for each of the set of (N-1) contingencies. If N = N_{lines}, the number of lines to be taken out of service, then the set of N_{lines} vectors $\left[\mathbf{d^{(n)}} \right]$ can be combined into a single $N_{buses} \times N_{lines}$ dense block:

$$\left[\begin{array}{c|c|c|c|c} \mathbf{d^{(1)}} & \mathbf{d^{(2)}} & \mathbf{d^{(3)}} & \dots & \mathbf{d^{(N_{lines})}} \end{array} \right] \tag{5.1}$$

Then the following multiplication produces another $N_{buses} \times N_{lines}$ dense block:

$$\left[\begin{array}{c|c|c|c} \mathbf{dS^{(1)}} & \mathbf{dS^{(2)}} & \dots & \mathbf{dS^{(N_{lines})}} \end{array} \right] = \left[\begin{array}{c} \mathbf{S} \end{array} \right] \cdot \left[\begin{array}{c|c|c|c} \mathbf{d^{(1)}} & \mathbf{d^{(2)}} & \dots & \mathbf{d^{(N_{lines})}} \end{array} \right] \tag{5.2}$$

Computing the above with one thread per dense vector, the CUDA broadcast functionality for the GPGPU can be used to broadcast values from $\left[\begin{array}{c} \mathbf{S} \end{array} \right]$ simultaneously to the vectors $\left[\begin{array}{c|c|c|c|c} \mathbf{d^{(1)}} & \mathbf{d^{(2)}} & \mathbf{d^{(3)}} & \dots & \mathbf{d^{(N_{lines})}} \end{array} \right]$ dramatically reducing memory accesses and memory access contention.

There are a number of approaches to optimizing the matrix-vector multiplication in the above computation, including a number of approaches for exploiting the sparsity of the matrix. However, it is not the purpose of this work to develop sparse matrix methods for GPGPU architecture. Accordingly, the routines and methods developed in this chapter do not specify the exact method used to represent the sparse matrix or carry out the computation above.

5.1.2 Polar to Rectangular Conversion of a Block of Dense Vectors

Equations of the form:

$$\left[\mathbf{V_i'^{(n)}} \right] = \left[\mathbf{V_i^{(n)}} \right] *. \left(\cos . \left[\theta_\mathbf{i}^{\mathbf{(n)}} \right] \right) \tag{5.3}$$

$$\left[\mathbf{V_i''^{(n)}} \right] = \left[\mathbf{V_i^{(n)}} \right] *. \left(\sin . \left[\theta_\mathbf{i}^{\mathbf{(n)}} \right] \right) \tag{5.4}$$

appeared repeatedly in Chapter 4, and polar to rectangular conversion of a block of dense vectors is a repeated task in the methods outlined in this thesis. Unlike the parts of the methods in which a sparse matrix multiplied is by a block of dense vectors – in which the same sparse matrix is needed across all (N-1) contingencies – polar to rectangular conversion of a block of dense vectors operates only upon data which is unique to each contingency and is fundamentally a question of sheer bulk processing.

Rather than than clog the too-few and too-slow special function units that perform the GPGPU sine and cosine functions, the proposed method is to use Taylor series approximations for $\cos\theta$ and $\sin\theta$:

$$\begin{aligned}\cos\theta &= \ (1 - 0.5 * \theta * \theta) \\ \sin\theta &= \left(\theta - \frac{\theta * \theta * \theta}{6}\right)\end{aligned} \tag{5.5}$$

To save on indexing and improve memory performance, $\left[\ \mathbf{V_i'^{(n)}}\ \right]$ and $\left[\ \mathbf{V_i''^{(n)}}\ \right]$ are not loaded and stored as separate vectors, but as a single complex vector of CUDA aligned type `float2`, such that

$$\begin{aligned}\left[\ \mathbf{V}', \mathbf{V_i''^{(n)}}\ \right][\ell].V' &= \left[\ \mathbf{V_i'^{(n)}}\ \right][\ell] \\ \left[\ \mathbf{V}', \mathbf{V_i''^{(n)}}\ \right][\ell].V'' &= \left[\ \mathbf{V_i''^{(n)}}\ \right][\ell]\end{aligned} \tag{5.6}$$

However, since $\left[\ \mathbf{V_i^{(n)}}\ \right]$ and $\left[\ \theta_\mathbf{i}^{(\mathbf{n})}\ \right]$ are not always needed at the same time, they

are loaded and stored as separate scalar vectors so that one or both are loaded as needed.

For a single `float2` vector $\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right]$, computed from a single pair of scalar vectors $\left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{n})}\ \right]$ and $\left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right]$,

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].V' &= \left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{n})}\ \right][\ell] \\
\theta &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].V' * \left(\theta - \frac{\theta * \theta * \theta}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].V' - 0.5 * \theta * \theta
\end{aligned} \tag{5.7}$$

for all ℓ.

While most of the routines in this chapter will not, as an example the details for this routine include tiling the blocks of dense vectors if the entire set is too large to fit in GPGPU memory at one time. The end goal of this routine is a complete set of vectors $\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right]$ in GPGPU memory for use in computing the current equations.

Computation of the Polar to Rectangular Conversion of a Block of Dense Vectors

1. Allocate and load the first set of vectors

$$\left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{3})}\ \middle|\ \cdots\ \right] \tag{5.8}$$

to fast memory on the GPGPU.

2. Allocate and load the first set of vectors

$$\left[\; \theta_{\mathbf{i}}^{(\mathbf{1})} \;\middle|\; \theta_{\mathbf{i}}^{(\mathbf{2})} \;\middle|\; \theta_{\mathbf{i}}^{(\mathbf{3})} \;\middle|\; \ldots \; \right] \tag{5.9}$$

to fast memory on the GPGPU.

3. Allocate the first set of `float2` vectors

$$\left[\; \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})} \;\middle|\; \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} \;\middle|\; \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{3})} \;\middle|\; \ldots \; \right] \tag{5.10}$$

to fast memory on the GPGPU.

4. While computation begins, allocate and load the second sets of vectors

$$\left[\; \mathbf{V}_{\mathbf{i}}^{(\mathbf{j})} \;\middle|\; \mathbf{V}_{\mathbf{i}}^{(\mathbf{k})} \;\middle|\; \ldots \; \right] \tag{5.11}$$

$$\left[\; \theta_{\mathbf{i}}^{(\mathbf{j})} \;\middle|\; \theta_{\mathbf{i}}^{(\mathbf{k})} \;\middle|\; \ldots \; \right] \tag{5.12}$$

and allocate the second set of vectors

$$\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{j})}\ \middle|\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \middle|\ \ldots\ \right] \tag{5.13}$$

5. Using one thread per $\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right]$ vector, compute iteratively for each ℓ:

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' &= \left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{1})}\ \right][\ell] \\
\theta^{(\ell,1)} &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' * \left(\theta^{(\ell,1)} - \frac{\theta^{(\ell,1)} * \theta^{(\ell,1)} * \theta^{(\ell,1)}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' - 0.5 * \theta^{(\ell,1)} * \theta^{(\ell,1)}
\end{aligned} \tag{5.14}$$

simultaneously with:

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})}\ \right][\ell].V' &= \left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{2})}\ \right][\ell] \\
\theta^{(\ell,2)} &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})}\ \right][\ell].V' * \left(\theta^{(\ell,2)} - \frac{\theta^{(\ell,2)} * \theta^{(\ell,2)} * \theta^{(\ell,2)}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})}\ \right][\ell].V' - 0.5 * \theta^{(\ell,2)} * \theta^{(\ell,2)} \\
&\vdots
\end{aligned} \tag{5.15}$$

6. When the first set of vectors completes computation, write out the first set of vectors if need be

$$\left[\begin{array}{c|c|c} \mathbf{V}', \mathbf{V_i''^{(1)}} & \mathbf{V}', \mathbf{V_i''^{(2)}} & \cdots \end{array}\right] \tag{5.16}$$

to the computer system main memory and to storage for future use. If there is room, skip this step and retain all $\left[\mathbf{V}', \mathbf{V_i''^{(n)}} \right]$ in GPGPU memory for use in computing the current equations.

7. Begin computation on the second set of vectors: Using one thread per $\left[\mathbf{V}', \mathbf{V_i''^{(n)}} \right]$ vector, compute iteratively for each ℓ:

$$\begin{aligned} \left[\mathbf{V}', \mathbf{V_i''^{(j)}} \right][\ell].V' &= \left[\mathbf{V_i^{(j)}} \right][\ell] \\ \theta^{(\ell,j)} &= \left[\theta_{\mathbf{i}}^{(\mathbf{j})} \right][\ell] \\ \left[\mathbf{V}', \mathbf{V_i''^{(j)}} \right][\ell].V'' &= \left[\mathbf{V}', \mathbf{V_i''^{(j)}} \right][\ell].V' * \left(\theta^{(\ell,j)} - \frac{\theta^{(\ell,j)} * \theta^{(\ell,j)} * \theta^{(\ell,j)}}{6} \right) \\ \left[\mathbf{V}', \mathbf{V_i''^{(j)}} \right][\ell].V' &= \left[\mathbf{V}', \mathbf{V_i''^{(j)}} \right][\ell].V' - 0.5 * \theta^{(\ell,j)} * \theta^{(\ell,j)} \end{aligned} \tag{5.17}$$

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \right][\ell].V' &= \left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{k})}\ \right][\ell] \\
\theta^{(\ell,k)} &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{k})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \right][\ell].V' * \left(\theta^{(\ell,k)} - \frac{\theta^{(\ell,k)} * \theta^{(\ell,k)} * \theta^{(\ell,k)}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \right][\ell].V' - 0.5 * \theta^{(\ell,k)} * \theta^{(\ell,k)} \\
&\vdots
\end{aligned} \tag{5.18}$$

8. If needed, load the third set of vectors $\left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{u})}\ \middle|\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{v})}\ \middle|\ \ldots\ \right]$ to the locations for the first set $\left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \ldots\ \right]$ and the third set of vectors $\left[\ \theta_{\mathbf{i}}^{(\mathbf{u})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{v})}\ \middle|\ \ldots\ \right]$ to the locations for the first set $\left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \ldots\ \right]$ while computation of the second set completes.

9. When the second set of vectors completes computation, write out the second set of vectors if need be

$$\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{j})}\ \middle|\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{k})}\ \middle|\ \ldots\ \right] \tag{5.19}$$

to the computer system main memory and to storage for future use. If there is room, skip this step and retain all $\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right]$ in GPGPU memory for use in computing the current equations.

10. Continue to interleave writing out the results of the last set of vectors and loading the next set of vectors with computing the current set until all have been computed.

11. Free the memory locations for the vectors
$\left[\begin{array}{c|c|c} \mathbf{V_i^{(1)}} & \mathbf{V_i^{(2)}} & \cdots \end{array}\right]$ and $\left[\begin{array}{c|c|c} \mathbf{V_i^{(j)}} & \mathbf{V_i^{(k)}} & \cdots \end{array}\right]$.

12. Free the memory locations for the vectors
$\left[\begin{array}{c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots \end{array}\right]$ and $\left[\begin{array}{c|c|c} \theta_{\mathbf{i}}^{(\mathbf{j})} & \theta_{\mathbf{i}}^{(\mathbf{k})} & \cdots \end{array}\right]$.

13. The variables $\theta^{(\ell,n)}$ should be thread-local register variables that are freed when their threads complete.

14. Retain $\left[\begin{array}{c} \mathbf{V}', \mathbf{V_i''^{(n)}} \end{array}\right]$ in GPGPU memory for use in computing the current equations, and load any $\left[\begin{array}{c} \mathbf{V}', \mathbf{V_i''^{(n)}} \end{array}\right]$ that have been written out to preserve room in GPGPU memory.

5.1.3 Calculating the Bus Currents

Calculating the bus currents follows the same procedure for the $P - \theta$ Iteration or the $Q - V$ Iteration, with the one exception being the procedure for calculating the first set of currents from the solution to the (N-0) base case. This part of the computation involves routines in which the same data is needed across all (N-1) contingencies, such as a sparse matrix multiplied by a dense block, and routines in which the data to be worked upon is unique to each contingency in a manner similar to the computation in Section 5.1.2.

The initial current estimate is computed according to:

$$\left[\ \mathbf{I_i'^{(n)}}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V_i'}\ \right] - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V_i''}\ \right] \tag{5.20}$$

$$\left[\ \mathbf{I_i''^{(n)}}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V_i''}\ \right] + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V_i'}\ \right] \tag{5.21}$$

The current adjustments to remove a line are as follows:

$$\begin{aligned} \mathbf{I'^{(n)}_{adjust,i}} = & \left[\ \mathbf{G_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V_i'^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_i'^{(n)}}\ \right][m] \right) \\ & - \left[\ \mathbf{B_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V_i''^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_i''^{(n)}}\ \right][m] \right) \end{aligned} \tag{5.22}$$

$$\begin{aligned} \left[\ \mathbf{I_i'^{(n)}}\ \right][\ell] &= \left[\ \mathbf{I_i'^{(n)}}\ \right][\ell] + \mathbf{I'^{(n)}_{adjust,i}} \\ \left[\ \mathbf{I_i'^{(n)}}\ \right][m] &= \left[\ \mathbf{I_i'^{(n)}}\ \right][m] - \mathbf{I'^{(n)}_{adjust,i}} \end{aligned} \tag{5.23}$$

$$\begin{aligned} \mathbf{I''^{(n)}_{adjust,i}} = & \left[\ \mathbf{G_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V_i''^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_i''^{(n)}}\ \right][m] \right) \\ & + \left[\ \mathbf{B_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V_i'^{(n)}}\ \right][\ell] - \left[\ \mathbf{V_i'^{(n)}}\ \right][m] \right) \end{aligned} \tag{5.24}$$

$$\begin{aligned} \left[\ \mathbf{I_i''^{(n)}}\ \right][\ell] &= \left[\ \mathbf{I_i''^{(n)}}\ \right][\ell] + \mathbf{I''^{(n)}_{adjust,i}} \\ \left[\ \mathbf{I_i''^{(n)}}\ \right][m] &= \left[\ \mathbf{I_i''^{(n)}}\ \right][m] - \mathbf{I''^{(n)}_{adjust,i}} \end{aligned} \tag{5.25}$$

$\left[\mathbf{I_i'^{(n)}} \right]$ and $\left[\mathbf{I_i''^{(n)}} \right]$ are stored as a single vector of CUDA aligned type `float2`: $\left[\mathbf{I', I_i''^{(n)}} \right]$.

Similarly, $\left[\mathbf{V_i'^{(n)}} \right]$ and $\left[\mathbf{V_i''^{(n)}} \right]$ are stored as a single vector of CUDA aligned type `float2`: $\left[\mathbf{V', V_i''^{(n)}} \right]$.

The end goal of this routine is a complete set of `float2` vectors

$$\left[\mathbf{I', I''^{(1)}} \middle| \mathbf{I', I''^{(2)}} \middle| \dots \middle| \mathbf{I', I''^{(N_{lines})}} \right] \tag{5.26}$$

and a complete set of `float2` vectors

$$\left[\mathbf{V', V''^{(1)}} \middle| \mathbf{V', V''^{(2)}} \middle| \dots \middle| \mathbf{V', V''^{(N_{lines})}} \right] \tag{5.27}$$

in GPGPU memory for use in computing the ΔP equation or the ΔQ equation.

Calculating the Bus Currents, Procedure

1. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\mathbf{G_{bus}} \right]$ and $\left[\mathbf{B_{bus}} \right]$
2. Allocate and load $\left[\mathbf{G_{bus}} \right]$ to the GPGPU according to the sparse-matrix-dense-block-multiplication routine chosen.
3. Allocate and load the set of `float2` vectors $\left[\mathbf{V', V_i''^{(1)}} \middle| \mathbf{V', V_i''^{(2)}} \middle| \dots \right]$ to fast memory on the GPGPU if they are not there already. The total block is $N_{buses} \times N_{lines}$.

4. Allocate the set of `float2` vectors $\left[\begin{array}{c|c|c} \mathbf{I}', \mathbf{I}''^{(1)}_{\mathbf{i}} & \mathbf{I}', \mathbf{I}''^{(2)}_{\mathbf{i}} & \cdots \end{array}\right]$ to fast memory on the GPGPU. The total block is $N_{buses} \times N_{lines}$.

5. Using the sparse-matrix-dense-block-multiplication routine chosen, simultaneously compute:

$$\left[\begin{array}{c|c|c} \mathbf{I}', \mathbf{I}''^{(1)}_{\mathbf{i}} & \mathbf{I}', \mathbf{I}''^{(2)}_{\mathbf{i}} & \cdots \end{array}\right] = \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \tag{5.28}$$

$$\cdot \left[\begin{array}{c|c|c} \mathbf{V}', \mathbf{V}''^{(1)}_{\mathbf{i}} & \mathbf{V}', \mathbf{V}''^{(2)}_{\mathbf{i}} & \cdots \end{array}\right] \tag{5.29}$$

which can also be written as

$$\begin{aligned}
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(1)}_{\mathbf{i}} \end{array}\right].I' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(1)}_{\mathbf{i}} \end{array}\right].V' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(1)}_{\mathbf{i}} \end{array}\right].I'' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(1)}_{\mathbf{i}} \end{array}\right].V'' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(2)}_{\mathbf{i}} \end{array}\right].I' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(2)}_{\mathbf{i}} \end{array}\right].V' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(2)}_{\mathbf{i}} \end{array}\right].I'' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(2)}_{\mathbf{i}} \end{array}\right].V'' \\
&\vdots
\end{aligned} \tag{5.30}$$

6. Free $\left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right]$.

7. Allocate and load $\left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right]$ to the GPGPU according to the sparse-matrix-dense-block-multiplication routine chosen.

8. Allocate and load the $N_{lines} \times 1$ vector $\left[\mathbf{G_{line}} \right]$.

9. Allocate and load the $N_{lines} \times 1$ vector $\left[\mathbf{B_{line}} \right]$.

10. Using the sparse-matrix-dense-block-multiplication routine chosen, simultaneously compute:

$$\begin{gathered}
\left[\mathbf{I}', \mathbf{I}''^{(\mathbf{1})}_{\mathbf{i}} \right].I' = \left[\mathbf{I}', \mathbf{I}''^{(\mathbf{1})}_{\mathbf{i}} \right].I' - \left[\mathbf{B_{bus}} \right] \cdot \left[\mathbf{V}', \mathbf{V}''^{(\mathbf{1})}_{\mathbf{i}} \right].V'' \\
\left[\mathbf{I}', \mathbf{I}''^{(\mathbf{1})}_{\mathbf{i}} \right].I'' = \left[\mathbf{I}', \mathbf{I}''^{(\mathbf{1})}_{\mathbf{i}} \right].I'' + \left[\mathbf{B_{bus}} \right] \cdot \left[\mathbf{V}', \mathbf{V}''^{(\mathbf{1})}_{\mathbf{i}} \right].V' \\
\left[\mathbf{I}', \mathbf{I}''^{(\mathbf{2})}_{\mathbf{i}} \right].I' = \left[\mathbf{I}', \mathbf{I}''^{(\mathbf{2})}_{\mathbf{i}} \right].I' - \left[\mathbf{B_{bus}} \right] \cdot \left[\mathbf{V}', \mathbf{V}''^{(\mathbf{2})}_{\mathbf{i}} \right].V'' \\
\left[\mathbf{I}', \mathbf{I}''^{(\mathbf{2})}_{\mathbf{i}} \right].I'' = \left[\mathbf{I}', \mathbf{I}''^{(\mathbf{2})}_{\mathbf{i}} \right].I'' + \left[\mathbf{B_{bus}} \right] \cdot \left[\mathbf{V}', \mathbf{V}''^{(\mathbf{2})}_{\mathbf{i}} \right].V' \\
\vdots
\end{gathered} \tag{5.31}$$

11. Free $\left[\mathbf{B_{bus}} \right]$.

12. Allocate an $N_{lines} \times 1$ vector of CUDA aligned type `float2`, $\left[\mathbf{I'_{adj}}, \mathbf{I''_{adj}} \right]$.

13. Using N_{lines} threads, for each single contingency n in which the line from bus ℓ to bus m is out of service, simultaneously compute:

$$\begin{aligned}
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj,i}}\ \right][n].I'_{adj} = & \left[\ \mathbf{G}_{\mathbf{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][\ell].V' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][m].V'\right) \\
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj,i}}\ \right][n].I''_{adj} = & \left[\ \mathbf{G}_{\mathbf{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][\ell].V'' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][m].V''\right) \\
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj,i}}\ \right][n].I'_{adj} = & \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj,i}}\ \right][n].I'_{adj} - \left[\ \mathbf{B}_{\mathbf{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][\ell].V'' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][m].V''\right) \\
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj,i}}\ \right][n].I''_{adj} = & \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj,i}}\ \right][n].I''_{adj} + \left[\ \mathbf{B}_{\mathbf{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][\ell].V' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{n})}_{\mathbf{i}}\ \right][m].V'\right)
\end{aligned} \tag{5.32}$$

for all n:

$$
\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][1].I'_{adj} &= \left[\ \mathbf{G_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][\ell_1].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][m_1].V' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][1].I''_{adj} &= \left[\ \mathbf{G_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][\ell_1].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][m_1].V'' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][1].I'_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][1].I'_{adj} - \left[\ \mathbf{B_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][\ell_1].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][m_1].V'' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][1].I''_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][1].I''_{adj} + \left[\ \mathbf{B_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][\ell_1].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}_{i}}\ \right][m_1].V' \right)
\end{aligned} \tag{5.33}
$$

$$
\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][2].I'_{adj} &= \left[\ \mathbf{G_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][\ell_2].V' - \left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][m_2].V'\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][2].I''_{adj} &= \left[\ \mathbf{G_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][\ell_2].V'' - \left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][m_2].V''\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][2].I'_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][2].I'_{adj} - \left[\ \mathbf{B_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][\ell_2].V'' - \left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][m_2].V''\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][2].I''_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj,i}}\ \right][2].I''_{adj} + \left[\ \mathbf{B_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][\ell_2].V' - \left[\ \mathbf{V'}, \mathbf{V_i''^{(2)}}\ \right][m_2].V'\right) \\
&\vdots
\end{aligned}
\tag{5.34}
$$

14. Free $\left[\ \mathbf{G_{line}}\ \right]$.

15. Free $\left[\ \mathbf{B_{line}}\ \right]$.

16. Using N_{lines} threads, simultaneously compute

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].I' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].I'' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][\ell].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][n].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][m].I' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][m].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][m].I'' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{n})}\ \right][m].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][n].I''_{adj}
\end{aligned} \tag{5.35}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell_1].I' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell_1].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell_1].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][1].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][m_1].I' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][m_1].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][m_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][m_1].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj},\mathbf{i}}\ \right][1].I''_{adj}
\end{aligned} \tag{5.36}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][\ell_2].I' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][\ell_2].I' + \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj,i}}\end{array}\right][2].I'_{adj} \quad (5.37)$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][\ell_2].I'' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][\ell_2].I'' + \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj,i}}\end{array}\right][2].I''_{adj}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][m_2].I' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][m_2].I' - \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj,i}}\end{array}\right][2].I'_{adj}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][m_2].I'' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}_{\mathbf{i}}''^{(\mathbf{2})}\end{array}\right][m_2].I'' - \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj,i}}\end{array}\right][2].I''_{adj}$$

$$\vdots \quad (5.38)$$

17. Free $\left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right]$.
18. Retain $\left[\begin{array}{c|c|c|c}\mathbf{I}',\mathbf{I}''^{(\mathbf{1})} & \mathbf{I}',\mathbf{I}''^{(\mathbf{2})} & \cdots & \mathbf{I}',\mathbf{I}''^{(\mathbf{N_{lines}})}\end{array}\right]$ in GPGPU memory for use in computing the ΔP equation or the ΔQ equation.
19. Retain $\left[\begin{array}{c|c|c|c}\mathbf{V}',\mathbf{V}''^{(\mathbf{1})} & \mathbf{V}',\mathbf{V}''^{(\mathbf{2})} & \cdots & \mathbf{V}',\mathbf{V}''^{(\mathbf{N_{lines}})}\end{array}\right]$ in GPGPU memory for use in computing the ΔP equation or the ΔQ equation.

5.2 Pre-Computation

This section details computations that can be performed outside the loops of the iterative FDPF solution algorithm.

5.2.1 The FDPF Base Case Constant Matrix Inverses

The matrices:

$$\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right],\ \left[\ \mathbf{B}''^{(\mathbf{0})}\ \right],\ \left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1},\text{ and }\left[\ \mathbf{B}''^{(\mathbf{0})}\ \right]^{-1} \tag{5.39}$$

are required to compute the set of $c_{B'}^{(n)}$ and $c_{B''}^{(n)}$ constants and the set of $\left[\ \mathbf{x_{B'}^{(n)}}\ \right]$ and $\left[\ \mathbf{x_{B''}^{(n)}}\ \right]$ vector constants. For the proposed method, $\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1}$, and $\left[\ \mathbf{B}''^{(\mathbf{0})}\ \right]^{-1}$ are computed on using an established CPU-based sparse matrix solver and then stored.

5.2.2 The $c_{B'}$ Constants and $x_{B'}$ Vectors

For a single contingency n in which the line from bus ℓ to bus m is out of service, $\left[\ \mathbf{cx_{B'}^{(n)}}\ \right]$ is calculated as follows:

$$b'_n = \left[\ \mathbf{B}'^{(\mathbf{0})}\ \right][\ell, m] \tag{5.40}$$

$$\left[\ \mathbf{x_{B'}^{(n)}}\ \right] = \left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1} \cdot \left[\ \mathbf{m^{(n)}}\ \right]^{\mathbf{T}} \tag{5.41}$$

$$c_{B'}^{(n)} = \left(\frac{1}{b'_n} + \left[\mathbf{m^{(n)}} \right] \cdot \left[\mathbf{B'^{(0)}} \right]^{-1} \cdot \left[\mathbf{m^{(n)}} \right]^{\mathbf{T}} \right)^{-1} \tag{5.42}$$
$$= \left(\frac{1}{b'_n} + \left[\mathbf{m^{(n)}} \right] \cdot \left[\mathbf{x_{B'}^{(n)}} \right] \right)^{-1}$$
$$= \left(\frac{1}{b'_n} + \left[\mathbf{x_{B'}^{(n)}} \right] [\ell] - \left[\mathbf{x_{B'}^{(n)}} \right] [m] \right)^{-1}$$

$$\left[\mathbf{cx_{B'}^{(n)}} \right] = c_{B'}^{(n)} \cdot \left[\mathbf{x_{B'}^{(n)}} \right] \tag{5.43}$$

The end goal of this routine is a complete set of vectors

$$\left[\mathbf{cx_{B'}^{(1)}} \middle| \mathbf{cx_{B'}^{(2)}} \middle| \cdots \middle| \mathbf{cx_{B'}^{(N_{lines})}} \right] \tag{5.44}$$

written out to computer system main memory and storage for use in the iterations of the FDPF contingency algorithm, if the version from Section 4.7 is being used. Otherwise, if the method from Section 4.6 is being used, the vectors $\left[\mathbf{cx_{B'}^{(n)}} \right]$ should be retained in GPGPU memory for use in computing The $\Delta\theta$ vectors.

However, if not all data needed for this routine will fit in GPGPU memory as it is needed, priority should be given to keeping the sparse matrices $\left[\mathbf{B'^{(0)}} \right]$ and $\left[\mathbf{B'^{(0)}} \right]^{-1}$ in GPGPU memory while swapping out sections of blocks of dense vectors, since it is the sparse matrices that provide data needed by all (N-1) contingencies.

5.2.3 Computation of the $c_{B'}$ Constants and $x_{B'}$ Vectors

1. Choose a sparse-matrix storage method based on the properties of $\left[\ \mathbf{B'^{(0)}}\ \right]$.

2. Allocate and load $\left[\ \mathbf{B'^{(0)}}\ \right]$ to texture memory according to the sparse-matrix storage method chosen.

3. Allocate an $N_{lines} \times 1$ vector $\left[\ \mathbf{c_{B'}}\ \right]$. This will eventually hold the set of $c_{B'}^{(n)}$ constants such that:

$$\left[\ \mathbf{c_{B'}}\ \right] = \left[\ \mathbf{c_{B'}^{(1)}} \middle| \mathbf{c_{B'}^{(2)}} \middle| \mathbf{c_{B'}^{(3)}} \middle| \cdots \middle| \mathbf{c_{B'}^{(N_{lines})}}\ \right] \tag{5.45}$$

4. Using N_{lines} threads, compute:

$$\left[\ \mathbf{c_{B'}}\ \right][n] = \frac{1}{\left[\ \mathbf{B'^{(0)}}\ \right][\ell, m]} \tag{5.46}$$

for all n:

$$\left[\ \mathbf{c_{B'}}\ \right][1] \left| \left[\ \mathbf{c_{B'}}\ \right][2] \right| \cdots = \frac{1}{\left[\ \mathbf{B'^{(0)}}\ \right][\ell 1, m1]} \left| \frac{1}{\left[\ \mathbf{B'^{(0)}}\ \right][\ell 2, m2]} \right| \cdots \tag{5.47}$$

5. Free $\left[\ \mathbf{B'^{(0)}}\ \right]$.

6. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1}$.

7. Allocate and load $\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1}$ according to the sparse-matrix-dense-block-multiplication routine chosen.

8. Allocate $\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{N_{lines}})}\ \right]$. The total block is $N_{buses} \times N_{lines}$.

9. Load $\left[\ \mathbf{m}^{(\mathbf{1}),\mathbf{T}}\ \middle|\ \mathbf{m}^{(\mathbf{2}),\mathbf{T}}\ \middle|\ \cdots\ \middle|\ \mathbf{m}^{(\mathbf{N_{lines}}),\mathbf{T}}\ \right]$ into $\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{N_{lines}})}\ \right]$.

10. Compute:

$$\begin{aligned}\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{N_{lines}})}\ \right] = &\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right] \\ &\cdot \left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{N_{lines}})}\ \right]\end{aligned} \tag{5.48}$$

11. Free $\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1}$.

12. Using N_{lines} threads, compute:

$$\left[\ \mathbf{c}_{\mathbf{B}'}\ \right][n] = \left(\left[\ \mathbf{c}_{\mathbf{B}'}\ \right][n] + \left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{n})}\ \right][\ell] - \left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{n})}\ \right][m]\right)^{-1} \tag{5.49}$$

and

$$\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{n})}\ \right][n] = \left(\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{n})}\ \right][n]\right) * \left(\left[\ \mathbf{c}_{\mathbf{B}'}\ \right][n]\right) \tag{5.50}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{c_{B'}}\ \right][1] &= \left(\left[\ \mathbf{c_{B'}}\ \right][1] + \left[\ \mathbf{cx_{B'}^{(1)}}\ \right][\ell 1] - \left[\ \mathbf{cx_{B'}^{(1)}}\ \right][m1]\right)^{-1} \\
\left[\ \mathbf{c_{B'}}\ \right][2] &= \left(\left[\ \mathbf{c_{B'}}\ \right][2] + \left[\ \mathbf{cx_{B'}^{(2)}}\ \right][\ell 2] - \left[\ \mathbf{cx_{B'}^{(2)}}\ \right][m2]\right)^{-1} \\
\left[\ \mathbf{c_{B'}}\ \right][3] &= \left(\left[\ \mathbf{c_{B'}}\ \right][3] + \left[\ \mathbf{cx_{B'}^{(3)}}\ \right][\ell 3] - \left[\ \mathbf{cx_{B'}^{(3)}}\ \right][m3]\right)^{-1} \\
&\vdots
\end{aligned} \tag{5.51}$$

and

$$\begin{aligned}
\left[\ \mathbf{cx_{B'}^{(1)}}\ \right][1] &= \left[\ \mathbf{cx_{B'}^{(1)}}\ \right][1]\left[\ \mathbf{c_{B'}}\ \right][1] \\
\left[\ \mathbf{cx_{B'}^{(2)}}\ \right][2] &= \left[\ \mathbf{cx_{B'}^{(2)}}\ \right][2]\left[\ \mathbf{c_{B'}}\ \right][2] \\
\left[\ \mathbf{cx_{B'}^{(3)}}\ \right][3] &= \left[\ \mathbf{cx_{B'}^{(3)}}\ \right][3]\left[\ \mathbf{c_{B'}}\ \right][3] \\
&\vdots
\end{aligned} \tag{5.52}$$

13. Free $\left[\ \mathbf{c_{B'}}\ \right]$.
14. Write out

$$\left[\ \mathbf{cx_{B'}^{(1)}} \;\middle|\; \mathbf{cx_{B'}^{(2)}} \;\middle|\; \cdots \;\middle|\; \mathbf{cx_{B'}^{(N_{lines})}}\ \right] \tag{5.53}$$

to the computer system main memory and to storage for future use.

15. If the method from Section 4.7 is being used, free

$$\left[\begin{array}{c|c|c|c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{1})} & \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{2})} & \cdots & \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.54}$$

Otherwise, if the method from Section 4.6 is being used, retain

$$\left[\begin{array}{c|c|c|c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{1})} & \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{2})} & \cdots & \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.55}$$

in GPGPU memory and move on to calculating

$$\left[\begin{array}{c|c|c|c} \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{1})} & \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{2})} & \cdots & \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.56}$$

5.2.4 The $\Delta\theta$ Vectors

As can be seen from Equation4.164, for a single contingency n in which the line from bus ℓ to bus m is out of service:

$$\begin{aligned} \left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] - c_{B'}^{(n)} \left[\begin{array}{c} \mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \end{array}\right] \\ \cdot \left(\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [\ell] - \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [m] \right) \end{aligned} \tag{5.57}$$

for the method in Section 4.6, computing $\left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{n})} \end{array}\right]$ only requires data from the base case and precomputed data such as $c_{B'}^{(n)}$ and $\left[\begin{array}{c} \mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \end{array}\right]$. This means

$$\left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{n})} \end{array}\right], \left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{2}}^{(\mathbf{n})} \end{array}\right], \left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{3}}^{(\mathbf{n})} \end{array}\right], \cdots \tag{5.58}$$

can be pre-computed before starting the main iterations of the FPDF Contingency Algorithm. They are, in fact, identical:

$$\begin{aligned} \left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{n})} \end{array}\right] &= \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] - c_{B'}^{(n)} \left[\begin{array}{c} \mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \end{array}\right] \\ &\cdot \left(\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [\ell] - \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [m] \right) \\ \left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{2}}^{(\mathbf{n})} \end{array}\right] &= \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] - c_{B'}^{(n)} \left[\begin{array}{c} \mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \end{array}\right] \\ &\cdot \left(\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [\ell] - \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [m] \right) \\ \left[\begin{array}{c} \boldsymbol{\Delta}\theta_{\mathbf{3}}^{(\mathbf{n})} \end{array}\right] &= \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] - c_{B'}^{(n)} \left[\begin{array}{c} \mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \end{array}\right] \\ &\cdot \left(\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [\ell] - \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] [m] \right) \\ &\vdots \end{aligned} \tag{5.59}$$

$\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right]$ has been stored from the solution of the base case.

The end goal of this routine is a complete set of vectors

$$\left[\begin{array}{c|c|c|c} \boldsymbol{\Delta}\theta^{(\mathbf{1})} & \boldsymbol{\Delta}\theta^{(\mathbf{2})} & \cdots & \boldsymbol{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.60}$$

written out to computer system main memory and storage for use in the iterations of the FDPF contingency algorithm using the Stott and Alsac approximation described in Section 4.6.

5.2.5 Pre-computation of the $\Delta\theta$ Vectors

1. $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B'}^{(1)}} & \mathbf{cx_{B'}^{(2)}} & \cdots & \mathbf{cx_{B'}^{(N_{lines})}} \end{array}\right]$ is already in GPGPU memory from computation of the $c_{B'}$ constants and $x_{B'}$ vectors.

2. Allocate and load the $N_{buses} \times 1$ vector $\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right]$ into constant memory.

3. Allocate $\left[\begin{array}{c|c|c|c} \boldsymbol{\Delta}\theta^{(\mathbf{1})} & \boldsymbol{\Delta}\theta^{(\mathbf{2})} & \cdots & \boldsymbol{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right]$. The entire block is $N_{buses} \times N_{lines}$.

4. Use the CUDA broadcast functionality to initialize

$$\left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\Delta}\theta^{(\mathbf{0})} \end{array}\right] \tag{5.61}$$

for all n:

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta}\theta^{(\mathbf{1})} & \mathbf{\Delta}\theta^{(\mathbf{2})} & \ldots & \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right] = \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{0})} \end{array}\right] \tag{5.62}$$

5. Using N_{lines} threads, compute

$$\begin{aligned} \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{n})} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{n})} \end{array}\right] \\ \cdot \left(\left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{n})} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{n})} \end{array}\right][m]\right) \end{aligned} \tag{5.63}$$

for all n:

$$\begin{aligned} \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{1})} \end{array}\right] &= \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{1})} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{1})} \end{array}\right] \\ &\cdot \left(\left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{1})} \end{array}\right][\ell 1] - \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{1})} \end{array}\right][m1]\right) \\ \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{2})} \end{array}\right] &= \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{2})} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{2})} \end{array}\right] \\ &\cdot \left(\left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{2})} \end{array}\right][\ell 2] - \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{2})} \end{array}\right][m2]\right) \\ &\vdots \\ \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right] &= \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{N_{lines}})} \end{array}\right] \\ &\cdot \left(\left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right][\ell N_{lines}] - \left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right][mN_{lines}]\right) \end{aligned} \tag{5.64}$$

6. Write out

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta}\theta^{(\mathbf{1})} & \mathbf{\Delta}\theta^{(\mathbf{2})} & \dots & \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.65}$$

to the computer system main memory and to storage for future use.

7. Free $\left[\begin{array}{c|c|c|c} \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{1})} & \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{2})} & \dots & \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{N_{lines}})} \end{array}\right]$.

8. Free $\left[\begin{array}{c} \mathbf{\Delta}\theta^{(\mathbf{0})} \end{array}\right]$.

9. Free $\left[\begin{array}{c|c|c|c} \mathbf{\Delta}\theta^{(\mathbf{1})} & \mathbf{\Delta}\theta^{(\mathbf{2})} & \dots & \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array}\right]$.

5.2.6 The $c_{B''}$ Constants and $x_{B''}$ Vectors

For a single contingency n in which the line from bus ℓ to bus m is out of service:

$$b''_n = \left[\begin{array}{c} \mathbf{B''^{(0)}} \end{array}\right][\ell, m] \tag{5.66}$$

$$\left[\begin{array}{c} \mathbf{x}_{\mathbf{B''}}^{(\mathbf{n})} \end{array}\right] = \left[\begin{array}{c} \mathbf{B''^{(0)}} \end{array}\right]^{-1} \cdot \left[\begin{array}{c} \mathbf{m}^{(\mathbf{n})} \end{array}\right]^{\mathbf{T}} \tag{5.67}$$

$$c_{B''}^{(n)} = \left(\frac{1}{b''_n} + \left[\ \mathbf{m^{(n)}}\ \right] \cdot \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \mathbf{m^{(n)}}\ \right]^{\mathbf{T}} \right)^{-1} \tag{5.68}$$

$$= \left(\frac{1}{b''_n} + \left[\ \mathbf{m^{(n)}}\ \right] \cdot \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \right)^{-1}$$

$$= \left(\frac{1}{b''_n} + \left[\ \mathbf{x_{B''}^{(n)}}\ \right][\ell] - \left[\ \mathbf{x_{B''}^{(n)}}\ \right][m] \right)^{-1}$$

$$\left[\ \mathbf{cx_{B''}^{(n)}}\ \right] = c_{B''}^{(n)} \cdot \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \tag{5.69}$$

The end goal of this routine is a complete set of vectors

$$\left[\ \mathbf{cx_{B''}^{(1)}} \ \middle|\ \mathbf{cx_{B''}^{(2)}} \ \middle|\ \cdots \ \middle|\ \mathbf{cx_{B''}^{(N_{lines})}}\ \right] \tag{5.70}$$

written out to computer system main memory and storage for use in the iterations of the FDPF contingency algorithm, if the version from Section 4.7 is being used. Otherwise, if the method from Section 4.6 is being used, the vectors $\left[\ \mathbf{cx_{B''}^{(n)}}\ \right]$ should be retained in GPGPU memory for use in computing The ΔV vectors.

However, if not all data needed for this routine will fit in GPGPU memory as it is needed, priority should be given to keeping the sparse matrices $\left[\ \mathbf{B''^{(0)}}\ \right]$ and $\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}$ in GPGPU memory while swapping out sections of blocks of dense vectors, since it is the sparse matrices that provide data needed by all (N-1) contingencies.

5.2.7 Computation of the $c_{B''}$ Constants and $x_{B''}$ Vectors

1. Choose a sparse-matrix storage method based on the properties of $\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]$.

2. Allocate and load $\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]$ to texture memory according to the sparse-matrix storage method chosen.

3. Allocate an $N_{lines} \times 1$ vector $\left[\begin{array}{c}\mathbf{c_{B''}}\end{array}\right]$. This will eventually hold the set of $c_{B''}^{(n)}$ constants such that:

$$\left[\begin{array}{c}\mathbf{c_{B''}}\end{array}\right] = \left[\begin{array}{c|c|c|c|c}\mathbf{c_{B''}^{(1)}} & \mathbf{c_{B''}^{(2)}} & \mathbf{c_{B''}^{(3)}} & \cdots & \mathbf{c_{B''}^{(N_{lines})}}\end{array}\right] \tag{5.71}$$

4. Using N_{lines} threads, compute:

$$\left[\begin{array}{c}\mathbf{c_{B''}}\end{array}\right][n] = \frac{1}{\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right][\ell, m]} \tag{5.72}$$

for all n:

$$\left[\begin{array}{c}\mathbf{c_{B''}}\end{array}\right][1]\left|\left[\begin{array}{c}\mathbf{c_{B''}}\end{array}\right][2]\right|\cdots = \frac{1}{\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right][\ell 1, m1]}\left|\frac{1}{\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right][\ell 2, m2]}\right|\cdots \tag{5.73}$$

5. Free $\left[\begin{array}{c}\mathbf{B''^{(0)}}\end{array}\right]$.

6. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\begin{array}{c} \mathbf{B''^{(0)}} \end{array}\right]^{-1}$.

7. Load $\left[\begin{array}{c} \mathbf{B''^{(0)}} \end{array}\right]^{-1}$ according to the sparse-matrix-dense-block-multiplication routine chosen.

8. Allocate $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$ or its first subset according to the sparse-matrix-dense-block-multiplication routine chosen. The total block is $N_{buses} \times N_{lines}$.

9. Load $\left[\begin{array}{c|c|c|c} \mathbf{m^{(1),T}} & \mathbf{m^{(2),T}} & \cdots & \mathbf{m^{(N_{lines}),T}} \end{array}\right]$ into $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$.

10. Compute:

$$\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right] = \left[\begin{array}{c} \mathbf{B''^{(0)}} \end{array}\right] \cdot \left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right] \tag{5.74}$$

11. Free $\left[\begin{array}{c} \mathbf{B''^{(0)}} \end{array}\right]^{-1}$.

12. Using N_{lines} threads, compute:

$$\left[\begin{array}{c} \mathbf{c_{B''}} \end{array}\right][n] = \left(\left[\begin{array}{c} \mathbf{c_{B''}} \end{array}\right][n] + \left[\begin{array}{c} \mathbf{cx_{B''}^{(n)}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(n)}} \end{array}\right][m]\right)^{-1} \tag{5.75}$$

and

$$\left[\begin{array}{c} \mathbf{cx_{B''}^{(n)}} \end{array}\right][n] = \left(\left[\begin{array}{c} \mathbf{cx_{B''}^{(n)}} \end{array}\right][n]\right) * \left(\left[\begin{array}{c} \mathbf{c_{B''}} \end{array}\right][n]\right) \tag{5.76}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{c_{B''}}\ \right][1] &= \left(\left[\ \mathbf{c_{B''}}\ \right][1] + \left[\ \mathbf{cx_{B''}^{(1)}}\ \right][\ell 1] - \left[\ \mathbf{cx_{B''}^{(1)}}\ \right][m1]\right)^{-1} \\
\left[\ \mathbf{c_{B''}}\ \right][2] &= \left(\left[\ \mathbf{c_{B''}}\ \right][2] + \left[\ \mathbf{cx_{B''}^{(2)}}\ \right][\ell 2] - \left[\ \mathbf{cx_{B''}^{(2)}}\ \right][m2]\right)^{-1} \\
\left[\ \mathbf{c_{B''}}\ \right][3] &= \left(\left[\ \mathbf{c_{B''}}\ \right][3] + \left[\ \mathbf{cx_{B''}^{(3)}}\ \right][\ell 3] - \left[\ \mathbf{cx_{B''}^{(3)}}\ \right][m3]\right)^{-1} \\
&\vdots
\end{aligned} \tag{5.77}$$

and

$$\begin{aligned}
\left[\ \mathbf{cx_{B''}^{(1)}}\ \right][1] &= \left[\ \mathbf{cx_{B''}^{(1)}}\ \right][1]\left[\ \mathbf{c_{B''}}\ \right][1] \\
\left[\ \mathbf{cx_{B''}^{(2)}}\ \right][2] &= \left[\ \mathbf{cx_{B''}^{(2)}}\ \right][2]\left[\ \mathbf{c_{B''}}\ \right][2] \\
\left[\ \mathbf{cx_{B''}^{(3)}}\ \right][3] &= \left[\ \mathbf{cx_{B'}^{(3)}}\ \right][3]\left[\ \mathbf{c_{B''}}\ \right][3] \\
&\vdots
\end{aligned} \tag{5.78}$$

13. Free $\left[\ \mathbf{c_{B''}}\ \right]$.

14. Write out

$$\left[\ \mathbf{cx_{B''}^{(1)}} \ \middle|\ \mathbf{cx_{B''}^{(2)}} \ \middle|\ \cdots \ \middle|\ \mathbf{cx_{B''}^{(N_{lines})}}\ \right] \tag{5.79}$$

to the computer system main memory and to storage for future use.

15. If the method from Section 4.7 is being used, free

$$\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right] \tag{5.80}$$

Otherwise, if the method from Section 4.6 is being used, retain

$$\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right] \tag{5.81}$$

in GPGPU memory and move on to calculating

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta V_1^{(1)}} & \mathbf{\Delta V_1^{(2)}} & \cdots & \mathbf{\Delta V_1^{(N_{lines})}} \end{array}\right] \tag{5.82}$$

5.2.8 The ΔV Vectors

As can be seen from Equation4.164, for a single contingency n in which the line from bus ℓ to bus m is out of service:

$$\begin{aligned} \left[\, \mathbf{\Delta V_1^{(n)}} \,\right] = \left[\, \mathbf{\Delta V^{(0)}} \,\right] - c_{B''}^{(n)} \left[\, \mathbf{x_{B''}^{(n)}} \,\right] \\ \cdot \left(\left[\, \mathbf{\Delta V^{(0)}} \,\right][\ell] - \left[\, \mathbf{\Delta V^{(0)}} \,\right][m] \right) \end{aligned} \tag{5.83}$$

for the method in Section 4.6, computing $\left[\ \mathbf{\Delta V_1^{(n)}}\ \right]$ only requires data from the base case and precomputed data such as $c_{B''}^{(n)}$ and $\left[\ \mathbf{x_{B''}^{(n)}}\ \right]$. This means

$$\left[\ \mathbf{\Delta V_1^{(n)}}\ \right], \left[\ \mathbf{\Delta V_2^{(n)}}\ \right], \left[\ \mathbf{\Delta V_3^{(n)}}\ \right], \cdots \tag{5.84}$$

can be pre-computed before starting the main iterations of the FPDF Contingency Algorithm. They are, in fact, identical:

$$\begin{aligned}
\left[\ \mathbf{\Delta V_1^{(n)}}\ \right] &= \left[\ \mathbf{\Delta V^{(0)}}\ \right] - c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \\
&\cdot \left(\left[\ \mathbf{\Delta V^{(0)}}\ \right][\ell] - \left[\ \mathbf{\Delta V^{(0)}}\ \right][m] \right) \\
\left[\ \mathbf{\Delta V_2^{(n)}}\ \right] &= \left[\ \mathbf{\Delta V^{(0)}}\ \right] - c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \\
&\cdot \left(\left[\ \mathbf{\Delta V^{(0)}}\ \right][\ell] - \left[\ \mathbf{\Delta V^{(0)}}\ \right][m] \right) \\
\left[\ \mathbf{\Delta V_3^{(n)}}\ \right] &= \left[\ \mathbf{\Delta V^{(0)}}\ \right] - c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \\
&\cdot \left(\left[\ \mathbf{\Delta V^{(0)}}\ \right][\ell] - \left[\ \mathbf{\Delta V^{(0)}}\ \right][m] \right) \\
&\vdots
\end{aligned} \tag{5.85}$$

$\left[\ \mathbf{\Delta V^{(0)}}\ \right]$ has been stored from the solution of the base case.

The end goal of this routine is a complete set of vectors

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta V^{(1)}} & \mathbf{\Delta V^{(2)}} & \dots & \mathbf{\Delta V^{(N_{lines})}} \end{array}\right] \tag{5.86}$$

written out to computer system main memory and storage for use in the iterations of the FDPF contingency algorithm using the Stott and Alsac approximation described in Section 4.6.

5.2.9 Pre-computation of the ΔV Vectors

1. $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \dots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$ is already in GPGPU memory from computation of the $c_{B''}$ constants and $x_{B''}$ vectors.

2. Allocate and load the $N_{buses} \times 1$ vector $\left[\begin{array}{c} \mathbf{\Delta V^{(0)}} \end{array}\right]$ into constant memory.

3. Allocate $\left[\begin{array}{c|c|c|c} \mathbf{\Delta V^{(1)}} & \mathbf{\Delta V^{(2)}} & \dots & \mathbf{\Delta V^{(N_{lines})}} \end{array}\right]$. The entire block is $N_{buses} \times N_{lines}$.

4. Use the CUDA broadcast functionality to initialize

$$\left[\begin{array}{c} \mathbf{\Delta V^{(n)}} \end{array}\right] = \left[\begin{array}{c} \mathbf{\Delta V^{(0)}} \end{array}\right] \tag{5.87}$$

for all n:

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta V^{(1)}} & \mathbf{\Delta V^{(2)}} & \dots & \mathbf{\Delta V^{(N_{lines})}} \end{array}\right] = \left[\begin{array}{c} \mathbf{\Delta V^{(0)}} \end{array}\right] \tag{5.88}$$

5. Using N_{lines} threads, compute

$$\begin{aligned} \left[\begin{array}{c} \mathbf{\Delta V^{(n)}} \end{array}\right] = & \left[\begin{array}{c} \mathbf{\Delta V^{(n)}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(n)}} \end{array}\right] \\ & \cdot \left(\left[\begin{array}{c} \mathbf{\Delta V^{(n)}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{\Delta V^{(n)}} \end{array}\right][m]\right) \end{aligned} \tag{5.89}$$

for all n:

$$\begin{aligned} \left[\begin{array}{c} \mathbf{\Delta V^{(1)}} \end{array}\right] = & \left[\begin{array}{c} \mathbf{\Delta V^{(1)}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(1)}} \end{array}\right] \\ & \cdot \left(\left[\begin{array}{c} \mathbf{\Delta V^{(1)}} \end{array}\right][\ell 1] - \left[\begin{array}{c} \mathbf{\Delta V^{(1)}} \end{array}\right][m1]\right) \\ \left[\begin{array}{c} \mathbf{\Delta V^{(2)}} \end{array}\right] = & \left[\begin{array}{c} \mathbf{\Delta V^{(2)}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(2)}} \end{array}\right] \\ & \cdot \left(\left[\begin{array}{c} \mathbf{\Delta V^{(2)}} \end{array}\right][\ell 2] - \left[\begin{array}{c} \mathbf{\Delta V^{(2)}} \end{array}\right][m2]\right) \\ & \vdots \\ \left[\begin{array}{c} \mathbf{\Delta V^{(N_{lines})}} \end{array}\right] = & \left[\begin{array}{c} \mathbf{\Delta V^{(N_{lines})}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right] \\ & \cdot \left(\left[\begin{array}{c} \mathbf{\Delta V^{(N_{lines})}} \end{array}\right][\ell N_{lines}] - \left[\begin{array}{c} \mathbf{\Delta V^{(N_{lines})}} \end{array}\right][m N_{lines}]\right) \end{aligned} \tag{5.90}$$

6. Write out

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta V^{(1)}} & \mathbf{\Delta V^{(2)}} & \cdots & \mathbf{\Delta V^{(N_{lines})}} \end{array}\right] \tag{5.91}$$

to the computer system main memory and to storage for future use.

7. Free $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$.

8. Free $\left[\begin{array}{c} \mathbf{\Delta V^{(0)}} \end{array}\right]$.

9. Free $\left[\begin{array}{c|c|c|c} \mathbf{\Delta V^{(1)}} & \mathbf{\Delta V^{(2)}} & \cdots & \mathbf{\Delta V^{(N_{lines})}} \end{array}\right]$.

5.3 FPDF (N-1) Contingencies Algorithm

This section will walk through computing the set of (N-1) contingencies for a power system with N_{lines} possible single outages. For this section, it is presumed that the base case for no lines out of services has been solved and the data available and that certain values have been pre-computed.

This section follows the method outlined in Section 4.7 for a single contingency n in which the line from bus ℓ to bus m is out of service and the full version of the $\Delta\theta$ and ΔV calculations are used. Where the method differs for the version in Section 4.6 that uses the Stott and Alsac approximation, the changes for the alternative method are noted and detailed.

5.3.1 Inputs to the FPDF (N-1) Contingencies Algorithm

1. $\left[\ \mathbf{V^{(0)}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar voltage magnitudes at each bus from the solution of the base case
2. $\left[\ \theta^{(\mathbf{0})}\ \right]$ is an $N_{buses} \times 1$ vector of scalar voltage angles at each bus from the solution of the base case
3. $\left[\ \mathbf{G_{bus}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of the real parts of the entries in the $\left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right]$ matrix for the base case
4. $\left[\ \mathbf{B_{bus}}\ \right]$ is an $N_{buses} \times N_{buses}$ matrix of the imaginary parts of the entries in the $\left[\ \vec{\mathbf{Y}}_{\mathbf{bus}}\ \right]$ matrix for the base case
5. $\left[\ \mathbf{P_{sched}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled active power at each bus
6. $\left[\ \mathbf{Q_{sched}}\ \right]$ is an $N_{buses} \times 1$ vector of scalar scheduled reactive power at each bus
7. $c_{B'}^{(n)}$ and $c_{B''}^{(n)}$ are pre-computed scalar constants for the $\mathbf{\Delta}\theta$ and $\mathbf{V}$ update equations
8. $\left[\ \mathbf{x_{B'}^{(n)}}\ \right]$ and $\left[\ \mathbf{x_{B''}^{(n)}}\ \right]$ pre-computed $N_{buses} \times 1$ vectors of scalar values for the $\mathbf{\Delta}\theta$ and $\mathbf{V}$ update equations

Items 7 through 8 above are different for the method outlined in Section 4.6 that uses the Stott and Alsac approximation. For that version they are replaced with:

1. $\left[\ \mathbf{\Delta}\theta^{(\mathbf{1})}\ \middle|\ \mathbf{\Delta}\theta^{(\mathbf{2})}\ \middle|\ \ldots\ \middle|\ \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})}\ \right]$ is a set of N_{lines} pre-computed $N_{buses} \times 1$ vectors for the θ update equation

2. $\left[\begin{array}{c|c|c|c} \mathbf{\Delta V^{(1)}} & \mathbf{\Delta V^{(2)}} & \ldots & \mathbf{\Delta V^{(N_{lines})}} \end{array} \right]$ is a set of N_{lines} pre-computed $N_{buses} \times 1$ vectors for the $\mathbf{V}$ update equation

5.3.2 The $P - \theta$ Iteration: First Iteration

The initial bus voltages for each of the set of (N-1) contingencies are equal to the bus voltages for the (N-0) base case. As a result, the first iteration can exploit greater re-use of data than the later iterations.

Converting the Initial Bus Voltages from Polar to Rectangular

The initial bus voltage magnitude vectors are all equal to the voltage magnitude vector from the solution of the base (N-0) case, $\left[\begin{array}{c} \mathbf{V^{(0)}} \end{array} \right]$:

$$\left[\begin{array}{c|c|c|c} \mathbf{V_0^{(1)}} & \mathbf{V_0^{(2)}} & \ldots & \mathbf{V_0^{(N_{lines})}} \end{array} \right] = \left[\begin{array}{c} \mathbf{V^{(0)}} \end{array} \right] \tag{5.92}$$

Similarly, the initial bus angle vectors are all equal to the voltage vector from the solution of the base (N-0) case, $\left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right]$:

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{0}}^{(\mathbf{1})} & \theta_{\mathbf{0}}^{(\mathbf{2})} & \ldots & \theta_{\mathbf{0}}^{(\mathbf{N_{lines}})} \end{array} \right] = \left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right] \tag{5.93}$$

It is not until the first set of current corrections that the vectors for the different contingencies n begin to diverge. This makes converting the initial voltages from polar to rectangular very simple, as it only has to be done for one $N_{buses} \times 1$ vector.

Following a reduced form of the procedure in Section 5.1.2, $\left[\ \mathbf{V}''^{(\mathbf{0})}\ \right]$ are also stored as a single vector of CUDA aligned type `float2`:
$\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right]$

The end goal of this routine is the vector $\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right]$ in GPGPU memory for the initial current calculations.

Converting the Initial Bus Voltages from Polar to Rectangular, Procedure

1. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\ \mathbf{V}^{(\mathbf{0})}\ \right]$ to texture memory on the GPGPU.

2. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\ \theta^{(\mathbf{0})}\ \right]$ to texture memory on the GPGPU.

3. Allocate the `float2` vector $\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right]$ to fast memory on the GPGPU.

4. Using one thread per vector entry $\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][\ell]$, compute:

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][1].V' &= \left[\ \mathbf{V}^{(\mathbf{0})}\ \right][1] \\
\theta_{0,1} &= \left[\ \theta^{(\mathbf{0})}\ \right][1] \\
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][1].V'' &= \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][1].V' * \left(\theta_{0,1} - \frac{\theta_{0,1} * \theta_{0,1} * \theta_{0,1}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][1].V' &= \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][1].V' - 0.5 * \theta_{0,1} * \theta_{0,1}
\end{aligned} \tag{5.94}$$

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][2].V' &= \left[\ \mathbf{V}^{(\mathbf{0})}\ \right][2] \\
\theta_{0,2} &= \left[\ \theta^{(\mathbf{0})}\ \right][2] \\
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][2].V'' &= \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][2].V' * \left(\theta_{0,2} - \frac{\theta_{0,2} * \theta_{0,2} * \theta_{0,2}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][2].V' &= \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][2].V' - 0.5 * \theta_{0,2} * \theta_{0,2} \\
&\vdots
\end{aligned} \tag{5.95}$$

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][N^{lines}].V' &= \left[\ \mathbf{V}^{(\mathbf{0})}\ \right][N^{lines}] \\
\theta_{0,N_{lines}} &= \left[\ \theta^{(\mathbf{0})}\ \right][N^{lines}] \\
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][N^{lines}].V'' &= \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][N^{lines}].V' \\
&* \left(\theta_{0,N_{lines}} - \frac{\theta_{0,N_{lines}} * \theta_{0,N_{lines}} * \theta_{0,N_{lines}}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][N^{lines}].V' &= \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right][N^{lines}].V' \\
&- 0.5 * \theta_{0,N_{lines}} * \theta_{0,N_{lines}}
\end{aligned} \tag{5.96}$$

5. Leave $\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{0})}\ \right]$ in GPGPU memory for the initial current calculations.
6. Free $\left[\ \mathbf{V}^{(\mathbf{0})}\ \right]$.
7. Free $\left[\ \theta^{(\mathbf{0})}\ \right]$.
8. The individual θ variables should be thread-local register variables that are freed when their threads complete.

Calculating the Initial Bus Currents

The initial bus current vectors, before they are corrected for the individual contingencies, are all identical, calculated from:

$$\left[\ \mathbf{I}'^{(0)}_{1/2}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}'^{(0)}_{\mathbf{0}}\ \right] - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}''^{(0)}_{\mathbf{0}}\ \right] \tag{5.97}$$

$$\left[\ \mathbf{I}''^{(0)}_{1/2}\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}''^{(0)}_{\mathbf{0}}\ \right] + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}'^{(0)}_{\mathbf{0}}\ \right] \tag{5.98}$$

It is not until the first set of current corrections that the vectors for the different contingencies n begin to diverge.

$\left[\ \mathbf{I}'^{(0)}\ \right]$ and $\left[\ \mathbf{I}''^{(0)}\ \right]$ are stored as a single vector of CUDA aligned type `float2`: $\left[\ \mathbf{I}', \mathbf{I}''^{(0)}\ \right]$.

The end goal of this routine is the to have the vectors $\left[\ \mathbf{V}', \mathbf{V}''^{(0)}\ \right]$, $\left[\ \mathbf{I}', \mathbf{I}''^{(0)}\ \right]$, $\left[\ \mathbf{G_{line}}\ \right]$ and $\left[\ \mathbf{B_{line}}\ \right]$ in GPGPU memory for use in computing the current corrections.

However, if not all data needed for this routine will fit in GPGPU memory as it is needed, priority should be given to keeping the sparse matrices $\left[\ \mathbf{G_{bus}}\ \right]$ and $\left[\ \mathbf{B_{bus}}\ \right]$ in GPGPU memory while swapping out sections of blocks of dense vectors, since it is the sparse matrices that provide data needed by all (N-1) contingencies.

Calculating the Initial Bus Currents, Procedure

1. The vector $\left[\ \mathbf{V}', \mathbf{V}''^{(0)}\ \right]$ is already on the GPGPU.
2. Allocate the `float2` vector $\left[\ \mathbf{I}', \mathbf{I}''^{(0)}\ \right]$ to fast memory on the GPGPU.
3. Choose a sparse-matrix-dense-vector-multiplication routine based on the properties of $\left[\ \mathbf{G_{bus}}\ \right]$.

4. Allocate and load $\left[\mathbf{G_{bus}} \right]$ to the GPGPU according to the sparse-matrix-dense-vector-multiplication routine chosen.

5. Using the sparse-matrix-dense-vector-multiplication routine chosen, compute:

$$\begin{aligned} \left[\mathbf{I}', \mathbf{I}''^{(\mathbf{0})} \right].I' &= \left[\mathbf{G_{bus}} \right] \cdot \left[\mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \right].V' \\ \left[\mathbf{I}', \mathbf{I}''^{(\mathbf{0})} \right].I'' &= \left[\mathbf{G_{bus}} \right] \cdot \left[\mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \right].V'' \end{aligned} \tag{5.99}$$

6. Allocate an $N_{lines} \times 1$ vector $\left[\mathbf{G_{line}} \right]$.

7. Using N_{lines} threads, for each single contingency n in which the line from bus ℓ to bus m is out of service, store:

$$\left[\mathbf{G_{line}} \right][n] = \left[\mathbf{G_{bus}} \right][l, m] \tag{5.100}$$

8. Write out $\left[\mathbf{G_{line}} \right]$ to computer system main memory and storage for future use, but also retain it in GPGPU memory.

9. Free $\left[\mathbf{G_{bus}} \right]$.

10. Choose a sparse-matrix-dense-vector-multiplication routine based on the properties of $\left[\mathbf{B_{bus}} \right]$.

11. Allocate and load $\left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right]$ to the GPGPU according to the sparse-matrix-dense-vector-multiplication routine chosen.

12. Using the sparse-matrix-dense-vector-multiplication routine chosen, compute:

$$\begin{aligned}\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{0})}\end{array}\right].I' &= \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{0})}\end{array}\right].I' - \left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right]\cdot\left[\begin{array}{c}\mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\end{array}\right].V'' \\ \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{0})}\end{array}\right].I'' &= \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{0})}\end{array}\right].I'' + \left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right]\cdot\left[\begin{array}{c}\mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\end{array}\right].V'\end{aligned} \tag{5.101}$$

13. Allocate an $N_{lines} \times 1$ vector $\left[\begin{array}{c}\mathbf{B_{line}}\end{array}\right]$.

14. Using N_{lines} threads, for each single contingency n in which the line from bus ℓ to bus m is out of service, store:

$$\left[\begin{array}{c}\mathbf{B_{line}}\end{array}\right][n] = \left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right][l,m] \tag{5.102}$$

15. Write out $\left[\begin{array}{c}\mathbf{B_{line}}\end{array}\right]$ to computer system main memory and storage for future use, but also retain it in GPGPU memory.

16. Free $\left[\begin{array}{c}\mathbf{B_{bus}}\end{array}\right]$.

17. Retain $\left[\begin{array}{c}\mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\end{array}\right]$ in GPGPU memory for the current corrections.

18. Retain $\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{0})}\end{array}\right]$ in GPGPU memory for the current corrections.

19. Retain $\left[\ \mathbf{G_{line}}\ \right]$ and $\left[\ \mathbf{B_{line}}\ \right]$ in GPGPU memory for the current corrections.

Calculating the Initial Bus Current Corrections

The current adjustments to remove a line are as follows:

$$\begin{aligned} \mathbf{I'^{(n)}_{adjust,1/2}} = & \left[\ \mathbf{G_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V'^{(n)}_0}\ \right][\ell] - \left[\ \mathbf{V'^{(n)}_0}\ \right][m] \right) \\ & - \left[\ \mathbf{B_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V''^{(n)}_0}\ \right][\ell] - \left[\ \mathbf{V''^{(n)}_0}\ \right][m] \right) \end{aligned} \quad (5.103)$$

$$\begin{aligned} \left[\ \mathbf{I'^{(n)}_{1/2}}\ \right][\ell] &= \left[\ \mathbf{I'^{(n)}_{1/2}}\ \right][\ell] + \mathbf{I'^{(n)}_{adjust,1/2}} \\ \left[\ \mathbf{I'^{(n)}_{1/2}}\ \right][m] &= \left[\ \mathbf{I'^{(n)}_{1/2}}\ \right][m] - \mathbf{I'^{(n)}_{adjust,1/2}} \end{aligned} \quad (5.104)$$

$$\begin{aligned} \mathbf{I''^{(n)}_{adjust,1/2}} = & \left[\ \mathbf{G_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V''^{(n)}_0}\ \right][\ell] - \left[\ \mathbf{V''^{(n)}_0}\ \right][m] \right) \\ & + \left[\ \mathbf{B_{line}}\ \right][n] \cdot \left(\left[\ \mathbf{V'^{(n)}_0}\ \right][\ell] - \left[\ \mathbf{V'^{(n)}_0}\ \right][m] \right) \end{aligned} \quad (5.105)$$

$$\begin{aligned} \left[\ \mathbf{I''^{(n)}_{1/2}}\ \right][\ell] &= \left[\ \mathbf{I''^{(n)}_{1/2}}\ \right][\ell] + \mathbf{I''^{(n)}_{adjust,1/2}} \\ \left[\ \mathbf{I''^{(n)}_{1/2}}\ \right][m] &= \left[\ \mathbf{I''^{(n)}_{1/2}}\ \right][m] - \mathbf{I''^{(n)}_{adjust,1/2}} \end{aligned} \quad (5.106)$$

The end goal of this routine is the to have the vector $\left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array}\right]$ and the set of vectors $\left[\begin{array}{c|c|c|c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} & \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} & \cdots & \mathbf{I}', \mathbf{I}''^{(\mathbf{N_{lines}})} \end{array}\right]$ in GPGPU memory for use in computing the ΔP equation.

Calculating the Initial Bus Current Corrections, Procedure

1. The vector $\left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array}\right]$ is already in GPGPU memory.
2. The vector $\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(\mathbf{0})} \end{array}\right]$ is already in GPGPU memory.
3. The vectors $\left[\begin{array}{c} \mathbf{G_{line}} \end{array}\right]$ and $\left[\begin{array}{c} \mathbf{B_{line}} \end{array}\right]$ are already in GPGPU memory.
4. Allocate an $N_{lines} \times 1$ vector of CUDA aligned type `float2`, $\left[\begin{array}{c} \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \end{array}\right]$.
5. Using N_{lines} threads, for each single contingency n in which the line from bus ℓ to bus m is out of service, simultaneously compute:

$$\begin{aligned} \left[\begin{array}{c} \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \end{array}\right][n].I'_{adj} &= \left[\begin{array}{c} \mathbf{G_{line}} \end{array}\right][n] \\ &\cdot \left(\left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array}\right][\ell].V' - \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array}\right][m].V'\right) \\ \left[\begin{array}{c} \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \end{array}\right][n].I''_{adj} &= \left[\begin{array}{c} \mathbf{G_{line}} \end{array}\right][n] \\ &\cdot \left(\left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array}\right][\ell].V'' - \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array}\right][m].V''\right) \end{aligned} \tag{5.107}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I'_{adj} = &\left[\ \mathbf{G_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][\ell_1].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][m_1].V'\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I''_{adj} = &\left[\ \mathbf{G_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][\ell_1].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][m_1].V''\right)
\end{aligned} \tag{5.108}$$

$$\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I'_{adj} = &\left[\ \mathbf{G_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][\ell_2].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][m_2].V'\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I''_{adj} = &\left[\ \mathbf{G_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][\ell_2].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(0)}}\ \right][m_2].V''\right) \\
&\vdots
\end{aligned} \tag{5.109}$$

6. Using N_{lines} threads simultaneously compute:

$$\begin{aligned}\left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][n].I'_{adj} &= \left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][n].I'_{adj} - \left[\ \mathbf{B}_{\mathbf{line}}\ \right][n] \\ &\cdot \left(\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][\ell].V'' - \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][m].V''\right) \\ \left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][n].I''_{adj} &= \left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][n].I''_{adj} + \left[\ \mathbf{B}_{\mathbf{line}}\ \right][n] \\ &\cdot \left(\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][\ell].V' - \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][m].V'\right)\end{aligned} \tag{5.110}$$

for all n:

$$\begin{aligned}\left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][1].I'_{adj} &= \left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][1].I'_{adj} - \left[\ \mathbf{B}_{\mathbf{line}}\ \right][1] \\ &\cdot \left(\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][\ell_1].V'' - \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][m_1].V''\right) \\ \left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][1].I''_{adj} &= \left[\ \mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\ \right][1].I''_{adj} + \left[\ \mathbf{B}_{\mathbf{line}}\ \right][1] \\ &\cdot \left(\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][\ell_1].V' - \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\ \right][m_1].V'\right)\end{aligned} \tag{5.111}$$

$$\begin{aligned}\left[\begin{array}{c}\mathbf{I'_{adj}},\mathbf{I''_{adj}}\end{array}\right][2].I'_{adj} &= \left[\begin{array}{c}\mathbf{I'_{adj}},\mathbf{I''_{adj}}\end{array}\right][2].I'_{adj} - \left[\begin{array}{c}\mathbf{B_{line}}\end{array}\right][2] \\ &\cdot \left(\left[\begin{array}{c}\mathbf{V'},\mathbf{V''^{(0)}}\end{array}\right][\ell_2].V'' - \left[\begin{array}{c}\mathbf{V'},\mathbf{V''^{(0)}}\end{array}\right][m_2].V''\right) \\ \left[\begin{array}{c}\mathbf{I'_{adj}},\mathbf{I''_{adj}}\end{array}\right][2].I''_{adj} &= \left[\begin{array}{c}\mathbf{I'_{adj}},\mathbf{I''_{adj}}\end{array}\right][2].I''_{adj} + \left[\begin{array}{c}\mathbf{B_{line}}\end{array}\right][2] \\ &\cdot \left(\left[\begin{array}{c}\mathbf{V'},\mathbf{V''^{(0)}}\end{array}\right][\ell_2].V' - \left[\begin{array}{c}\mathbf{V'},\mathbf{V''^{(0)}}\end{array}\right][m_2].V'\right) \\ &\vdots\end{aligned} \tag{5.112}$$

7. Free $\left[\begin{array}{c}\mathbf{B_{line}}\end{array}\right]$.

8. Allocate the set of `float2` vectors $\left[\begin{array}{c|c|c|c}\mathbf{I'},\mathbf{I''^{(1)}} & \mathbf{I'},\mathbf{I''^{(2)}} & \cdots & \mathbf{I'},\mathbf{I''^{(N_{lines})}}\end{array}\right]$. The total block is $N_{buses} \times N_{lines}$.

9. Use the CUDA broadcast functionality to initialize

$$\left[\begin{array}{c}\mathbf{I'},\mathbf{I''^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{I'},\mathbf{I''^{(0)}}\end{array}\right] \tag{5.113}$$

for all n:

$$\left[\begin{array}{c|c|c|c}\mathbf{I'},\mathbf{I''^{(1)}} & \mathbf{I'},\mathbf{I''^{(2)}} & \cdots & \mathbf{I'},\mathbf{I''^{(N_{lines})}}\end{array}\right] = \left[\begin{array}{c}\mathbf{I'},\mathbf{I''^{(0)}}\end{array}\right] \tag{5.114}$$

10. Free $\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{0})} \ \right]$.

11. Using N_{lines} threads, compute

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][\ell].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][\ell].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][\ell].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][\ell].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][n].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][m].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][m].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][m].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})} \ \right][m].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][n].I''_{adj}
\end{aligned} \tag{5.115}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][\ell_1].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][\ell_1].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][\ell_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][\ell_1].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][1].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][m_1].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][m_1].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][m_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \ \right][m_1].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}} \ \right][1].I''_{adj}
\end{aligned} \tag{5.116}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I' + \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][1].I'_{adj} \quad (5.117)$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I'' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I'' + \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][1].I''_{adj}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I' - \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][1].I'_{adj}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I'' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I'' - \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][1].I''_{adj}$$

$$\vdots$$

(5.118)

12. Free $\left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right]$.

13. Retain $\left[\begin{array}{c|c|c|c}\mathbf{I}',\mathbf{I}''^{(\mathbf{1})} & \mathbf{I}',\mathbf{I}''^{(\mathbf{2})} & \cdots & \mathbf{I}',\mathbf{I}''^{(\mathbf{N_{lines}})}\end{array}\right]$ in GPGPU memory for use in computing the ΔP equation.

14. Retain $\left[\begin{array}{c}\mathbf{V}',\mathbf{V}''^{(\mathbf{0})}\end{array}\right]$ in GPGPU memory for use in computing the ΔP equation.

The ΔP Equation, First Iteration

The equations to calculate $\left[\begin{array}{c}\frac{\mathbf{\Delta P_1^{(n)}}}{\mathbf{V_1^{(n)}}}\end{array}\right]$ are as follows:

$$\left[\begin{array}{c}\mathbf{P_1^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{V_0'^{(n)}}\end{array}\right] *. \left[\begin{array}{c}\mathbf{I_{1/2}'^{(n)}}\end{array}\right] + \left[\begin{array}{c}\mathbf{V_0''^{(n)}}\end{array}\right] *. \left[\begin{array}{c}\mathbf{I_{1/2}''^{(n)}}\end{array}\right] \quad (5.119)$$

$$\left[\begin{array}{c}\mathbf{\Delta P_1^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{P_{sched}}\end{array}\right] - \left[\begin{array}{c}\mathbf{P_1^{(n)}}\end{array}\right] \quad (5.120)$$

$$\Delta P_{max,1}^{(n)} = \max\left(\left[\begin{array}{c}\mathbf{\Delta P_1^{(n)}}\end{array}\right]\right) \tag{5.121}$$

$$\left[\begin{array}{c}\frac{\mathbf{\Delta P_1^{(n)}}}{\mathbf{V_1^{(n)}}}\end{array}\right] = \left[\begin{array}{c}\mathbf{\Delta P_1^{(n)}}\end{array}\right] /. \left[\begin{array}{c}\mathbf{V_0^{(n)}}\end{array}\right] \tag{5.122}$$

Where the calculation for this iteration differs from the later iterations is that the bus voltages have not changed from the solution of the base (N-0) case:

$$\left[\begin{array}{c|c|c|c}\mathbf{V}', \mathbf{V_0''^{(1)}} & \mathbf{V}', \mathbf{V_0''^{(2)}} & \cdots & \mathbf{V}', \mathbf{V_0''^{(N_{lines})}}\end{array}\right] = \left[\begin{array}{c}\mathbf{V}', \mathbf{V}''^{(0)}\end{array}\right] \tag{5.123}$$

The end goals of this routine are:

1. The vector $\left[\begin{array}{c}\mathbf{\Delta P_{max,1}}\end{array}\right]$ written out to computer system main memory to use to test for convergence.

2. A complete set of vectors

$$\left[\begin{array}{c|c|c}\mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \cdots\end{array}\right] \tag{5.124}$$

that have been normalized to

$$\left[\begin{array}{c|c|c} \frac{\mathbf{\Delta P^{(1)}}}{\mathbf{V^{(n)}}} & \frac{\mathbf{\Delta P^{(2)}}}{\mathbf{V^{(n)}}} & \dots \end{array} \right] \tag{5.125}$$

in GPGPU memory for use in calculating the θ equation.

The ΔP Equation, First Iteration, Procedure

1. $\left[\begin{array}{c|c|c|c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} & \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} & \dots & \mathbf{I}', \mathbf{I}''^{(\mathbf{N_{lines}})} \end{array} \right]$ is still in GPGPU memory.
2. $\left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array} \right]$ is still in GPGPU memory.
3. Use N_{lines} threads and the CUDA broadcast functionality to compute simultaneously
   :

$$\begin{aligned} \left[\begin{array}{c|c|c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} & \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} & \dots \end{array} \right] = & \left[\begin{array}{c|c|c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} & \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} & \dots \end{array} \right] \\ & * \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{0})} \end{array} \right] \end{aligned} \tag{5.126}$$

which can also be written:

$$\begin{aligned}
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(1)} \end{array}\right].I' &= \left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(1)} \end{array}\right].I' * . \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(0)} \end{array}\right].V' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(1)} \end{array}\right].I'' &= \left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(1)} \end{array}\right].I'' * . \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(0)} \end{array}\right].V'' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(2)} \end{array}\right].I' &= \left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(2)} \end{array}\right].I' * . \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(0)} \end{array}\right].V' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(2)} \end{array}\right].I'' &= \left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(2)} \end{array}\right].I'' * . \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(0)} \end{array}\right].V'' \\
&\vdots
\end{aligned} \tag{5.127}$$

4. Free $\left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(0)} \end{array}\right]$.

5. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\begin{array}{c} \mathbf{P_{sched}} \end{array}\right]$.

6. Allocate $\left[\begin{array}{c|c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \ldots & \mathbf{\Delta P^{(N_{lines})}} \end{array}\right]$. The entire block is $N_{buses} \times N_{lines}$.

7. Use N_{lines} threads and the CUDA broadcast functionality to initialize

$$\left[\begin{array}{c} \mathbf{\Delta P^{(n)}} \end{array}\right] = \left[\begin{array}{c} \mathbf{P_{sched}} \end{array}\right] \tag{5.128}$$

for all n:

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \ldots & \mathbf{\Delta P^{(N_{lines})}} \end{array}\right] = \left[\begin{array}{c} \mathbf{P_{sched}} \end{array}\right] \tag{5.129}$$

8. Free $\left[\ \mathbf{P_{sched}}\ \right]$.

9. Use N_{lines} threads to compute simultaneously

$$\begin{aligned} \left[\ \mathbf{\Delta P^{(n)}}\ \right] &= \left[\ \mathbf{\Delta P^{(n)}}\ \right] - \left[\ \mathbf{I', I''^{(n)}}\ \right].I' \\ \left[\ \mathbf{\Delta P^{(n)}}\ \right] &= \left[\ \mathbf{\Delta P^{(n)}}\ \right] - \left[\ \mathbf{I', I''^{(n)}}\ \right].I'' \end{aligned} \tag{5.130}$$

for all n:

$$\begin{aligned} \left[\ \mathbf{\Delta P^{(1)}}\ \right] &= \left[\ \mathbf{\Delta P^{(1)}}\ \right] - \left[\ \mathbf{I', I''^{(1)}}\ \right].I' \\ \left[\ \mathbf{\Delta P^{(1)}}\ \right] &= \left[\ \mathbf{\Delta P^{(1)}}\ \right] - \left[\ \mathbf{I', I''^{(1)}}\ \right].I'' \end{aligned} \tag{5.131}$$

$$\begin{aligned} \left[\ \mathbf{\Delta P^{(2)}}\ \right] &= \left[\ \mathbf{\Delta P^{(2)}}\ \right] - \left[\ \mathbf{I', I''^{(2)}}\ \right].I' \\ \left[\ \mathbf{\Delta P^{(2)}}\ \right] &= \left[\ \mathbf{\Delta P^{(2)}}\ \right] - \left[\ \mathbf{I', I''^{(2)}}\ \right].I'' \\ &\vdots \end{aligned} \tag{5.132}$$

10. Free $\left[\ \mathbf{I', I''^{(1)}}\ \middle|\ \mathbf{I', I''^{(2)}}\ \middle|\ \cdots\ \middle|\ \mathbf{I', I''^{(N_{lines})}}\ \right]$.

11. Allocate the $N_{lines} \times 1$ scalar vector $\left[\ \mathbf{\Delta P_{max,1}}\ \right]$.

12. Use N_{lines} threads to compute simultaneously

$$\Delta P_{max,1}[n] = \max\left(\left[\ \mathbf{\Delta P^{(n)}}\ \right]\right) \tag{5.133}$$

for all n:

$$\begin{aligned} \Delta P_{max,1}[1] &= \max\left(\left[\ \mathbf{\Delta P^{(1)}}\ \right]\right) \\ \Delta P_{max,1}[2] &= \max\left(\left[\ \mathbf{\Delta P^{(2)}}\ \right]\right) \\ &\vdots \end{aligned} \tag{5.134}$$

13. Write out $\left[\ \mathbf{\Delta P_{max,1}}\ \right]$ to the computer system main memory and storage to use to test for convergence.

14. Free $\left[\ \mathbf{\Delta P_{max,1}}\ \right]$.

15. Allocate and load the vector $\left[\ \mathbf{V^{(0)}}\ \right]$ to texture memory on the GPGPU.

16. Use N_{lines} threads and the CUDA broadcast functionality to compute simultaneously:

$$\left[\ \mathbf{\Delta P^{(n)}}\ \right] = \left[\ \mathbf{\Delta P^{(n)}}\ \right] ./ \left[\ \mathbf{V^{(0)}}\ \right] \tag{5.135}$$

for all n:

$$\left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \cdots\ \right] = \left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \cdots\ \right] /. \left[\ \mathbf{V^{(0)}}\ \right] \tag{5.136}$$

17. Free $\left[\ \mathbf{V^{(0)}}\ \right]$.

18. Retain $\left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \cdots\ \right]$ in GPGPU memory for use in calculating the θ equation.

The θ Equation, First Iteration, Full $\Delta\theta$

The updated of bus voltage angles $\left[\ \theta_{\mathbf{1}}^{\mathbf{(n)}}\ \right]$ are computed from:

$$\left[\ \mathbf{\Delta}\theta_{\mathbf{temp,1}}^{\mathbf{(n)}}\ \right] = \left[\ \mathbf{B'}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta P_1^{(n)}}}{\mathbf{V_1^{(n)}}}\ \right] \tag{5.137}$$

$$\begin{aligned}\left[\ \mathbf{\Delta}\theta_{\mathbf{1}}^{\mathbf{(n)}}\ \right] &= \left[\ \mathbf{\Delta}\theta_{\mathbf{temp,1}}^{\mathbf{(n)}}\ \right] - c_{B'}^{(n)} \left[\ \mathbf{x_{B'}^{(n)}}\ \right] \\ &\cdot \left(\left[\ \mathbf{\Delta}\theta_{\mathbf{temp,1}}^{\mathbf{(n)}}\ \right][\ell] - \left[\ \mathbf{\Delta}\theta_{\mathbf{temp,1}}^{\mathbf{(n)}}\ \right][m] \right)\end{aligned}$$

$$\left[\ \theta_{\mathbf{1}}^{\mathbf{(n)}}\ \right] = \left[\ \theta_{\mathbf{0}}^{\mathbf{(n)}}\ \right] + \left[\ \mathbf{\Delta}\theta_{\mathbf{1}}^{\mathbf{(n)}}\ \right] \tag{5.138}$$

Where the calculation for this iteration differs from the later iterations is that the bus voltages have not changed from the solution of the base (N-0) case:

$$\left[\theta_{\mathbf{1}}^{(\mathbf{1})} \middle| \theta_{\mathbf{1}}^{(\mathbf{2})} \middle| \ldots \middle| \theta^{(\mathbf{N_{lines}})} \right] = \left[\theta^{(\mathbf{0})} \right] \tag{5.139}$$

The end goal of this routine is a complete set of vectors

$$\left[\theta_{\mathbf{1}}^{(\mathbf{1})} \middle| \theta_{\mathbf{1}}^{(\mathbf{2})} \middle| \ldots \middle| \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \right] \tag{5.140}$$

in GPGPU memory for use in calculating the updated bus voltages and written out to computer system memory and storage for use in the next iteration.

However, if not all data needed for this routine will fit in GPGPU memory as it is needed, priority should be given to keeping the sparse matrix $\left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1}$ in GPGPU memory while swapping out sections of blocks of dense vectors, since it is the sparse matrix that provides data needed by all (N-1) contingencies.

The θ Equation, First Iteration, Procedure, Full $\Delta\theta$

1. $\left[\mathbf{\Delta P}^{(\mathbf{1})} \middle| \mathbf{\Delta P}^{(\mathbf{2})} \middle| \ldots \right]$ is still in GPGPU memory.
2. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1}$.
3. Allocate and load $\left[\mathbf{B}'^{(\mathbf{0})} \right]^{-1}$ according to the sparse-matrix-dense-block-multiplication routine chosen.
4. Allocate $\left[\theta_{\mathbf{1}}^{(\mathbf{1})} \middle| \theta_{\mathbf{1}}^{(\mathbf{2})} \middle| \ldots \middle| \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \right]$. The total block is $N_{buses} \times N_{lines}$.

5. Using the sparse-matrix-dense-block-multiplication routine chosen, compute:

$$\left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})}\ \right] = \left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1} \cdot \left[\ \mathbf{\Delta P}^{(\mathbf{1})}\ \middle|\ \mathbf{\Delta P}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{\Delta P}^{(\mathbf{N_{lines}})}\ \right] \tag{5.141}$$

6. Free $\left[\ \mathbf{B}'^{(\mathbf{0})}\ \right]^{-1}$.

7. Rename $\left[\ \mathbf{\Delta P}^{(\mathbf{1})}\ \middle|\ \mathbf{\Delta P}^{(\mathbf{2})}\ \middle|\ \cdots\ \right]$ to $\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{N_{lines}})}\ \right]$ and load $\left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{N_{lines}})}\ \right]$ to that location on the GPGPU. The total block is $N_{buses} \times N_{lines}$.

8. Using N_{lines} threads, compute simultaneously:

$$\left[\ \theta_{\mathbf{1}}^{(\mathbf{n})}\ \right] = \left[\ \theta_{\mathbf{1}}^{(\mathbf{n})}\ \right] - \left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{n})}\ \right] \cdot \left(\left[\ \theta_{\mathbf{1}}^{(\mathbf{n})}\ \right][\ell] - \left[\ \theta_{\mathbf{1}}^{(\mathbf{n})}\ \right][m] \right) \tag{5.142}$$

for all n:

$$\begin{aligned} \left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \right] &= \left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \right] - \left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{1})}\ \right] \cdot \left(\left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \right][\ell] - \left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \right][m] \right) \\ \left[\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \right] &= \left[\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \right] - \left[\ \mathbf{cx}_{\mathbf{B}'}^{(\mathbf{2})}\ \right] \cdot \left(\left[\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \right][\ell] - \left[\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \right][m] \right) \\ &\vdots \end{aligned} \tag{5.143}$$

9. Free $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B'}^{(1)}} & \mathbf{cx_{B'}^{(2)}} & \cdots & \mathbf{cx_{B'}^{(N_{lines})}} \end{array} \right]$.

10. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right]$.

11. Using N_{lines} threads and the CUDA broadcast functionality, compute simultaneously

$$\left[\begin{array}{c} \theta_{\mathbf{1}}^{(\mathbf{n})} \end{array} \right] = \left[\begin{array}{c} \theta_{\mathbf{1}}^{(\mathbf{n})} \end{array} \right] + \left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right] \tag{5.144}$$

for all n:

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array} \right] = \left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array} \right] + \left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right] \tag{5.145}$$

12. Free $\left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right]$.

13. Write out

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array} \right] \tag{5.146}$$

to computer system main memory for use in the next iteration, and retain it in GPGPU memory for use in calculating the updated bus voltages.

The θ Equation, First Iteration, Stott and Alsac Approximation

The updated of bus voltage angles $\left[\theta_{\mathbf{1}}^{(\mathbf{n})} \right]$ are computed from:

$$\left[\boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{n})} \right] = \left[\boldsymbol{\Delta}\theta^{(\mathbf{0})} \right] - c_{B'}^{(n)} \left[\mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \right] \cdot \left(\left[\boldsymbol{\Delta}\theta^{(\mathbf{0})} \right] [\ell] - \left[\boldsymbol{\Delta}\theta^{(\mathbf{0})} \right] [m] \right) \tag{5.147}$$

where the above has been pre-computed, and

$$\left[\theta_{\mathbf{1}}^{(\mathbf{n})} \right] = \left[\theta_{\mathbf{0}}^{(\mathbf{n})} \right] + \left[\boldsymbol{\Delta}\theta_{\mathbf{1}}^{(\mathbf{n})} \right] \tag{5.148}$$

Where the calculation for this iteration differs from the later iterations is that the bus voltages have not changed from the solution of the base (N-0) case:

$$\left[\theta_{\mathbf{1}}^{(\mathbf{1})} \middle| \theta_{\mathbf{1}}^{(\mathbf{2})} \middle| \cdots \middle| \theta^{(\mathbf{N_{lines}})} \right] = \left[\theta^{(\mathbf{0})} \right] \tag{5.149}$$

The end goal of this routine is a complete set of vectors

$$\left[\theta_{\mathbf{1}}^{(\mathbf{1})} \middle| \theta_{\mathbf{1}}^{(\mathbf{2})} \middle| \cdots \middle| \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \right] \tag{5.150}$$

in GPGPU memory for use in calculating the updated bus voltages and written out to computer system memory and storage for use in the next iteration.

The θ Equation, First Iteration, Procedure, Stott and Alsac Approximation

1. $\left[\begin{array}{c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \ldots \end{array} \right]$ is still in GPGPU memory.

2. Rename $\left[\begin{array}{c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \ldots \end{array} \right]$ to

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \ldots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array} \right] \tag{5.151}$$

   and load the set of N_{lines} pre-computed $N_{buses} \times 1$ vectors

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta}\theta^{(\mathbf{1})} & \mathbf{\Delta}\theta^{(\mathbf{2})} & \ldots & \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array} \right] \tag{5.152}$$

   to that location. The total block is $N_{buses} \times N_{lines}$.

3. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right]$.

4. Using N_{lines} threads and the CUDA broadcast functionality, compute simultaneously

$$\left[\begin{array}{c} \theta_{\mathbf{1}}^{(\mathbf{n})} \end{array} \right] = \left[\begin{array}{c} \theta_{\mathbf{1}}^{(\mathbf{n})} \end{array} \right] + \left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right] \tag{5.153}$$

   for all n:

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \ldots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array} \right] = \left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \ldots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array} \right] + \left[\begin{array}{c} \theta^{(\mathbf{0})} \end{array} \right] \tag{5.154}$$

5. Free $\left[\; \theta^{(\mathbf{0})} \; \right]$.

6. Write out

$$\left[\; \theta_{\mathbf{1}}^{(\mathbf{1})} \; \middle| \; \theta_{\mathbf{1}}^{(\mathbf{2})} \; \middle| \; \cdots \; \middle| \; \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \; \right] \tag{5.155}$$

to computer system main memory for use in the next iteration, and retain it in GPGPU memory for use in calculating the updated bus voltages.

Updating the Bus Voltages, First Iteration.

While there is now a distinct vector $\left[\; \theta_{\mathbf{1}}^{(\mathbf{n})} \; \right]$ for each contingency n, the bus voltage magnitudes have not yet been updated from the (N-0) base case, so that,

$$\left[\; \mathbf{V}_{\mathbf{0}}^{(\mathbf{1})} \; \middle| \; \mathbf{V}_{\mathbf{0}}^{(\mathbf{2})} \; \middle| \; \cdots \; \middle| \; \mathbf{V}_{\mathbf{0}}^{(\mathbf{N_{lines}})} \; \right] = \left[\; \mathbf{V}^{(\mathbf{0})} \; \right] \tag{5.156}$$

The end goal of this routine is the set of vectors

$$\left[\; \mathbf{V}', \mathbf{V}''^{(\mathbf{1})} \; \middle| \; \mathbf{V}', \mathbf{V}''^{(\mathbf{2})} \; \middle| \; \mathbf{V}', \mathbf{V}''^{(\mathbf{3})} \; \middle| \; \cdots \; \right] \tag{5.157}$$

in GPGPU memory for the current update calculations.

Updating the Bus Voltages, First Iteration, Procedure.

1. $\left[\begin{array}{c|c|c|c} \theta_{\mathbf{1}}^{(\mathbf{1})} & \theta_{\mathbf{1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})} \end{array}\right]$ is already in GPGPU memory.

2. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\begin{array}{c} \mathbf{V^{(0)}} \end{array}\right]$.

3. Allocate the set of `float2` vectors

$$\left[\begin{array}{c|c|c|c} \mathbf{V', V''^{(1)}} & \mathbf{V', V''^{(2)}} & \mathbf{V', V''^{(3)}} & \cdots \end{array}\right] \tag{5.158}$$

   to fast memory on the GPGPU.

4. Use N_{lines} threads and the CUDA broadcast functionality to initialize

$$\left[\begin{array}{c} \mathbf{V', V''^{(n)}} \end{array}\right].V' = \left[\begin{array}{c} \mathbf{V^{(0)}} \end{array}\right] \tag{5.159}$$

   for all n:

$$\begin{aligned} \left[\begin{array}{c} \mathbf{V', V''^{(1)}} \end{array}\right].V' &= \left[\begin{array}{c} \mathbf{V^{(0)}} \end{array}\right] \\ \left[\begin{array}{c} \mathbf{V', V''^{(2)}} \end{array}\right].V' &= \left[\begin{array}{c} \mathbf{V^{(0)}} \end{array}\right] \\ &\vdots \end{aligned} \tag{5.160}$$

5. Free $\left[\ \mathbf{V^{(0)}}\ \right]$.

6. Using one thread per $\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right]$ vector, compute iteratively for each ℓ:

$$
\begin{aligned}
\theta_1^{(\ell,1)} &= \left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \right][\ell] \\
\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell].V'' &= \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell].V' * \left(\theta_1^{(\ell,1)} - \frac{\theta_1^{(\ell,1)} * \theta_1^{(\ell,1)} * \theta_1^{(\ell,1)}}{6}\right) \\
\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell].V' &= \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell].V' - 0.5 * \theta_1^{(\ell,1)} * \theta_1^{(\ell,1)}
\end{aligned}
\tag{5.161}
$$

simultaneously with:

$$
\begin{aligned}
\theta_1^{(\ell,2)} &= \left[\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \right][\ell] \\
\left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell].V'' &= \left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell].V' * \left(\theta_1^{(\ell,2)} - \frac{\theta_1^{(\ell,2)} * \theta_1^{(\ell,2)} * \theta_1^{(\ell,2)}}{6}\right) \\
\left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell].V' &= \left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell].V' - 0.5 * \theta_1^{(\ell,2)} * \theta_1^{(\ell,2)} \\
&\vdots
\end{aligned}
\tag{5.162}
$$

7. Free $\left[\ \theta_{\mathbf{1}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{1}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{1}}^{(\mathbf{N_{lines}})}\ \right]$

8. Retain $\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \middle|\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \middle|\ \mathbf{V'}, \mathbf{V''^{(3)}}\ \middle|\ \cdots\ \right]$ in GPGPU memory for the current update calculations.

5.3.3 The $Q - V$ Iteration

The procedure is mainly the same for the $Q - V$ both during the first iteration and subsequent iterations. However, the initial bus voltages magnitudes for each of the set of (N-1) contingencies are equal to the bus voltage magnitudes for the (N-0) base case during the first iteration, but they are distinct for each contingency for later iterations. As a result, the later iterations are slightly more data-intensive than the first iteration. This section details the routines for the later iterations, with the steps that are different for the first iteration identified and detailed.

Calculating the Bus Currents

The set of `float2` vectors

$$\left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \middle|\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \middle|\ \cdots\ \right] \tag{5.163}$$

is already in GPGPU memory from the completion of the $P - \theta$ iteration.

The end goal of this routine is a complete set of `float2` vectors

$$\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{I}', \mathbf{I}''^{(\mathbf{N_{lines}})}\ \right] \tag{5.164}$$

and a complete set of `float2` vectors

$$\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{N_{lines}})}\ \right] \tag{5.165}$$

in GPGPU memory for use in computing the ΔQ equation.

The procedure is that detailed in Section 5.1.3.

Calculating the Bus Currents, Procedure

1. The set of `float2` vectors $\left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \middle|\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \middle|\ \ldots\ \right]$ are already in GPGPU memory from the completion of the $P-\theta$ iteration.

2. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\ \mathbf{G_{bus}}\ \right]$ and $\left[\ \mathbf{B_{bus}}\ \right]$

3. Allocate and load $\left[\ \mathbf{G_{bus}}\ \right]$ to the GPGPU according to the sparse-matrix-dense-block-multiplication routine chosen.

4. Allocate the set of `float2` vectors $\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \middle|\ \ldots\ \right]$ to fast memory on the GPGPU. The total block is $N_{buses} \times N_{lines}$.

5. Using the sparse-matrix-dense-block-multiplication routine chosen, simultaneously compute:

$$\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \middle|\ \ldots\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \tag{5.166}$$

$$\cdot \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \middle|\ \ldots\ \right] \tag{5.167}$$

which can also be written as

$$\begin{aligned}
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \end{array}\right].I' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{1})} \end{array}\right].V' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \end{array}\right].I'' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{1})} \end{array}\right].V'' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} \end{array}\right].I' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{2})} \end{array}\right].V' \\
\left[\begin{array}{c} \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} \end{array}\right].I'' &= \left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right] \cdot \left[\begin{array}{c} \mathbf{V}', \mathbf{V}''^{(\mathbf{2})} \end{array}\right].V'' \\
&\vdots
\end{aligned} \tag{5.168}$$

6. Free $\left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right]$.

7. Allocate and load $\left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right]$ to the GPGPU according to the sparse-matrix-dense-block-multiplication routine chosen.

8. Allocate and load the $N_{lines} \times 1$ vector $\left[\begin{array}{c} \mathbf{G_{line}} \end{array}\right]$.

9. Allocate and load the $N_{lines} \times 1$ vector $\left[\begin{array}{c} \mathbf{B_{line}} \end{array}\right]$.

10. Using the sparse-matrix-dense-block-multiplication routine chosen, simultaneously compute:

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right].I' - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{1})}\ \right].V'' \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right].I'' + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{1})}\ \right].V' \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right].I' - \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right].V'' \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right].I'' + \left[\ \mathbf{B_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right].V' \\
&\vdots
\end{aligned} \tag{5.169}$$

11. Free $\left[\ \mathbf{B_{bus}}\ \right]$.

12. Allocate an $N_{lines} \times 1$ vector of CUDA aligned type `float2`, $\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right]$.

13. Using N_{lines} threads, for each single contingency n in which the line from bus ℓ to bus m is out of service, simultaneously compute:

$$
\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I'_{adj} = & \left[\ \mathbf{G_{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V'\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I''_{adj} = & \left[\ \mathbf{G_{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V''\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I'_{adj} = & \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I'_{adj} - \left[\ \mathbf{B_{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V''\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I''_{adj} = & \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I''_{adj} + \left[\ \mathbf{B_{line}}\ \right][n] \\
& \cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V'\right)
\end{aligned}
\tag{5.170}
$$

for all n:

$$
\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I'_{adj} &= \left[\ \mathbf{G_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell_1].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][m_1].V' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I''_{adj} &= \left[\ \mathbf{G_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell_1].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][m_1].V'' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I'_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I'_{adj} - \left[\ \mathbf{B_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell_1].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][m_1].V'' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I''_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][1].I''_{adj} + \left[\ \mathbf{B_{line}}\ \right][1] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][\ell_1].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \right][m_1].V' \right)
\end{aligned}
\tag{5.171}
$$

$$
\begin{aligned}
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I'_{adj} &= \left[\ \mathbf{G}_{\mathbf{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][\ell_2].V' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][m_2].V'\right) \\
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I''_{adj} &= \left[\ \mathbf{G}_{\mathbf{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][\ell_2].V'' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][m_2].V''\right) \\
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I'_{adj} &= \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I'_{adj} - \left[\ \mathbf{B}_{\mathbf{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][\ell_2].V'' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][m_2].V''\right) \\
\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I''_{adj} &= \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I''_{adj} + \left[\ \mathbf{B}_{\mathbf{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][\ell_2].V' - \left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \right][m_2].V'\right) \\
&\vdots
\end{aligned} \tag{5.172}
$$

14. Free $\left[\ \mathbf{G}_{\mathbf{line}}\ \right]$.

15. Free $\left[\ \mathbf{B}_{\mathbf{line}}\ \right]$.

16. Using N_{lines} threads, simultaneously compute

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I''_{adj}
\end{aligned} \tag{5.173}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I''_{adj}
\end{aligned} \tag{5.174}$$

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][\ell_2].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][\ell_2].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I'_{adj} \quad (5.175)\\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][\ell_2].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][\ell_2].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I''_{adj}\\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][m_2].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][m_2].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I'_{adj}\\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][m_2].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \right][m_2].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][2].I''_{adj}\\
&\vdots
\end{aligned} \tag{5.176}$$

17. Free $\left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right]$.
18. Retain $\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}', \mathbf{I}''^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{I}', \mathbf{I}''^{(\mathbf{N_{lines}})}\ \right]$ in GPGPU memory for use in computing the ΔQ equation.
19. Retain $\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{N_{lines}})}\ \right]$ in GPGPU memory for use in computing the ΔQ equation.

The ΔQ Equation

The calculations for the ΔQ equation are as follows:

$$\left[\ \mathbf{Q_1^{(n)}}\ \right] = \left[\ \mathbf{V''^{(n)}_{1/2}}\ \right] *. \left[\ \mathbf{I_1'^{(n)}}\ \right] - \left[\ \mathbf{V'^{(n)}_{1/2}}\ \right] *. \left[\ \mathbf{I_1''^{(n)}}\ \right] \tag{5.177}$$

$$\left[\ \mathbf{\Delta Q_1^{(n)}}\ \right] = \left[\ \mathbf{Q_{sched}}\ \right] - \left[\ \mathbf{Q_1^{(n)}}\ \right] \tag{5.178}$$

$$\Delta Q_{max,1}^{(n)} = \max\left(\left[\ \mathbf{\Delta Q_1^{(n)}}\ \right]\right) \tag{5.179}$$

$$\left[\ \frac{\mathbf{\Delta Q_1^{(n)}}}{\mathbf{V_1^{(n)}}}\ \right] = \left[\ \mathbf{\Delta Q_1^{(n)}}\ \right] /. \left[\ \mathbf{V_{1/2}^{(n)}}\ \right] \tag{5.180}$$

Where the calculation for later iterations differs from the first iteration is that the bus voltage magnitudes are distinct for the various (N-1) contingencies.

The end goals of this routine are:

1. The vector $\left[\ \mathbf{\Delta Q_{max,i}}\ \right]$ written out to computer system main memory to use to test for convergence.

2. A complete set of vectors

$$\left[\ \mathbf{\Delta Q^{(1)}}\ \middle|\ \mathbf{\Delta Q^{(2)}}\ \middle|\ \ldots\ \right] \tag{5.181}$$

in GPGPU memory for use in calculating the V equation.

The ΔQ Equation, Procedure

1. $\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \middle|\ \ldots\ \middle|\ \mathbf{I}',\mathbf{I}''^{(\mathbf{N_{lines}})}\ \right]$ is still in GPGPU memory.
2. $\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \middle|\ \ldots\ \middle|\ \mathbf{V}',\mathbf{V}''^{(\mathbf{N_{lines}})}\ \right]$ is still in GPGPU memory.
3. Use N_{lines} threads to compute simultaneously :

$$\begin{aligned}\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \middle|\ \ldots\ \right] = & \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \middle|\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \middle|\ \ldots\ \right] \\ & *.\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \middle|\ \ldots\ \right]\end{aligned} \tag{5.182}$$

which can also be written:

$$\begin{aligned}\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I' *. \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \right].V' \\ \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I'' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I'' *. \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \right].V'' \\ \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I' *. \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \right].V' \\ \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I'' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I'' *. \left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \right].V'' \\ &\vdots\end{aligned} \tag{5.183}$$

4. Free $\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \middle|\ \ldots\ \middle|\ \mathbf{V}',\mathbf{V}''^{(\mathbf{N_{lines}})}\ \right]$.

5. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\mathbf{Q_{sched}} \right]$.

6. Use N_{lines} threads and the CUDA broadcast functionality to initialize

$$\left[\mathbf{\Delta Q^{(n)}} \right] = \left[\mathbf{Q_{sched}} \right] \tag{5.184}$$

for all n:

$$\left[\mathbf{\Delta Q^{(1)}} \middle| \mathbf{\Delta Q^{(2)}} \middle| \ldots \middle| \mathbf{\Delta Q^{(N_{lines})}} \right] = \left[\mathbf{Q_{sched}} \right] \tag{5.185}$$

7. Free $\left[\mathbf{Q_{sched}} \right]$.

8. Use N_{lines} threads to compute simultaneously

$$\begin{aligned} \left[\mathbf{\Delta Q^{(n)}} \right] &= \left[\mathbf{\Delta Q^{(n)}} \right] - \left[\mathbf{I'}, \mathbf{I''^{(n)}} \right] .I' \\ \left[\mathbf{\Delta Q^{(n)}} \right] &= \left[\mathbf{\Delta Q^{(n)}} \right] + \left[\mathbf{I'}, \mathbf{I''^{(n)}} \right] .I'' \end{aligned} \tag{5.186}$$

for all n:

$$\begin{bmatrix} \mathbf{\Delta Q^{(1)}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Delta Q^{(1)}} \end{bmatrix} - \begin{bmatrix} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \end{bmatrix} .I' \tag{5.187}$$
$$\begin{bmatrix} \mathbf{\Delta Q^{(1)}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Delta Q^{(1)}} \end{bmatrix} + \begin{bmatrix} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \end{bmatrix} .I''$$

$$\begin{bmatrix} \mathbf{\Delta Q^{(2)}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Delta Q^{(2)}} \end{bmatrix} - \begin{bmatrix} \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} \end{bmatrix} .I' \tag{5.188}$$
$$\begin{bmatrix} \mathbf{\Delta Q^{(2)}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Delta Q^{(2)}} \end{bmatrix} + \begin{bmatrix} \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} \end{bmatrix} .I''$$
$$\vdots$$

9. Free $\left[\mathbf{I}', \mathbf{I}''^{(\mathbf{1})} \middle| \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} \middle| \cdots \middle| \mathbf{I}', \mathbf{I}''^{(\mathbf{N_{lines}})} \right]$.

10. Allocate the $N_{lines} \times 1$ scalar vector $\begin{bmatrix} \mathbf{\Delta Q_{max,i}} \end{bmatrix}$.

11. Use N_{lines} threads to compute simultaneously

$$\Delta Q_{max,i}[n] = \max\left(\begin{bmatrix} \mathbf{\Delta Q^{(n)}} \end{bmatrix}\right) \tag{5.189}$$

for all n:

$$\Delta Q_{max,i}[1] = \max\left(\left[\begin{array}{c}\mathbf{\Delta Q^{(1)}}\end{array}\right]\right) \tag{5.190}$$
$$\Delta Q_{max,i}[2] = \max\left(\left[\begin{array}{c}\mathbf{\Delta Q^{(2)}}\end{array}\right]\right)$$
$$\vdots$$

12. Write out $\left[\begin{array}{c}\mathbf{\Delta Q_{max,i}}\end{array}\right]$ to the computer system main memory and storage to use to test for convergence.

13. Free $\left[\begin{array}{c}\mathbf{\Delta Q_{max,i}}\end{array}\right]$.

14. Allocate and load $\left[\begin{array}{c|c|c|c}\mathbf{V^{(1)}} & \mathbf{V^{(2)}} & \ldots & \mathbf{V^{(N_{lines})}}\end{array}\right]$. The entire block is $N_{buses} \times N_{lines}$.

15. Use N_{lines} threads to compute simultaneously:

$$\left[\begin{array}{c}\mathbf{\Delta Q^{(n)}}\end{array}\right] = \left[\begin{array}{c}\mathbf{\Delta Q^{(n)}}\end{array}\right] ./ \left[\begin{array}{c}\mathbf{V^{(n)}}\end{array}\right] \tag{5.191}$$

for all n:

$$\left[\begin{array}{c|c|c}\mathbf{\Delta Q^{(1)}} & \mathbf{\Delta Q^{(2)}} & \ldots\end{array}\right] = \left[\begin{array}{c|c|c}\mathbf{\Delta Q^{(1)}} & \mathbf{\Delta Q^{(2)}} & \ldots\end{array}\right] ./ \left[\begin{array}{c|c|c}\mathbf{V^{(1)}} & \mathbf{V^{(2)}} & \ldots\end{array}\right] \tag{5.192}$$

16. Free $\left[\mathbf{V^{(1)}} \middle| \mathbf{V^{(2)}} \middle| \ldots \middle| \mathbf{V^{(N_{lines})}} \right]$.

17. Retain $\left[\mathbf{\Delta Q^{(1)}} \middle| \mathbf{\Delta Q^{(2)}} \middle| \ldots \right]$ in GPGPU memory for use in calculating the V equation.

Steps 14 through 16 above are different for the first iteration, since the bus voltage magnitudes in the first iteration are identical across the (N-1) contingencies and are equal to those from the solution of the base (N-0) case:

$$\left[\mathbf{V', V_0''^{(1)}} \middle| \mathbf{V', V_0''^{(2)}} \middle| \ldots \middle| \mathbf{V', V_0''^{(N_{lines})}} \right] = \left[\mathbf{V', V''^{(0)}} \right] \tag{5.193}$$

The version of those steps for the first iteration are as follows:

1. Allocate and load the vector $\left[\mathbf{V^{(0)}} \right]$ to texture memory on the GPGPU.

2. Use N_{lines} threads and the CUDA broadcast functionality to compute simultaneously:

$$\left[\mathbf{\Delta Q^{(n)}} \right] = \left[\mathbf{\Delta Q^{(n)}} \right] /. \left[\mathbf{V^{(0)}} \right] \tag{5.194}$$

for all n:

$$\left[\ \mathbf{\Delta Q^{(1)}}\ \middle|\ \mathbf{\Delta Q^{(2)}}\ \middle|\ \cdots\ \right] = \left[\ \mathbf{\Delta Q^{(1)}}\ \middle|\ \mathbf{\Delta Q^{(2)}}\ \middle|\ \cdots\ \right] /. \left[\ \mathbf{V^{(0)}}\ \right] \tag{5.195}$$

3. Free $\left[\ \mathbf{V^{(0)}}\ \right]$.

The V Equation, Full ΔV

The equations for calculating the updated bus voltage magnitudes are as follows:

$$\begin{aligned} \left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right] &= \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \cdot \left[\ \frac{\mathbf{\Delta Q^{(n)}}}{\mathbf{V^{(n)}}}\ \right] \\ \left[\ \mathbf{\Delta V^{(n)}}\ \right] &= \left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right] - c^{(n)}_{B''} \left[\ \mathbf{x^{(n)}_{B''}}\ \right] \\ &\quad \cdot \left(\left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right][\ell] - \left[\ \mathbf{\Delta V^{(n)}_{temp}}\ \right][m] \right) \end{aligned} \tag{5.196}$$

$$\left[\ \mathbf{V^{(n)}_1}\ \right] = \left[\ \mathbf{V^{(n)}_0}\ \right] + \left[\ \mathbf{\Delta V^{(n)}_1}\ \right] \tag{5.197}$$

The end goal of this routine is a complete set of vectors

$$\left[\ \mathbf{V^{(1)}_i}\ \middle|\ \mathbf{V^{(2)}_i}\ \middle|\ \cdots\ \middle|\ \mathbf{V^{(N_{lines})}_i}\ \right] \tag{5.198}$$

in GPGPU memory for use in calculating the updated bus voltages and written out to computer system memory and storage for use in the next iteration.

However, if not all data needed for this routine will fit in GPGPU memory as it is needed, priority should be given to keeping the sparse matrix $\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}$ in GPGPU memory while swapping out sections of blocks of dense vectors, since it is the sparse matrix that provides data needed by all (N-1) contingencies.

The V Equation, Procedure, Full ΔV

1. $\left[\ \mathbf{\Delta Q^{(1)}}\ \middle|\ \mathbf{\Delta Q^{(2)}}\ \middle|\ \dots\ \right]$ is still in GPGPU memory.
2. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}$.
3. Allocate and load $\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}$ according to the sparse-matrix-dense-block-multiplication routine chosen.
4. Allocate $\left[\ \mathbf{V_i^{(1)}}\ \middle|\ \mathbf{V_i^{(2)}}\ \middle|\ \dots\ \middle|\ \mathbf{V_i^{(N_{lines})}}\ \right]$. The total block is $N_{buses} \times N_{lines}$.
5. Using the sparse-matrix-dense-block-multiplication routine chosen, compute:

$$\begin{aligned} \left[\ \mathbf{V_i^{(1)}}\ \middle|\ \mathbf{V_i^{(2)}}\ \middle|\ \dots\ \middle|\ \mathbf{V_i^{(N_{lines})}}\ \right] = & \left[\ \mathbf{B''^{(0)}}\ \right]^{-1} \\ & \cdot \left[\ \mathbf{\Delta Q^{(1)}}\ \middle|\ \mathbf{\Delta Q^{(2)}}\ \middle|\ \dots\ \middle|\ \mathbf{\Delta Q^{(N_{lines})}}\ \right] \end{aligned} \tag{5.199}$$

6. Free $\left[\ \mathbf{B''^{(0)}}\ \right]^{-1}$.

7. Rename $\left[\begin{array}{c|c|c} \mathbf{\Delta Q^{(1)}} & \mathbf{\Delta Q^{(2)}} & \cdots \end{array}\right]$ to $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$ and load $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$ to that location on the GPGPU. The total block is $N_{buses} \times N_{lines}$.

8. Using N_{lines} threads, compute simultaneously:

$$\left[\begin{array}{c} \mathbf{V_i^{(n)}} \end{array}\right] = \left[\begin{array}{c} \mathbf{V_i^{(n)}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(n)}} \end{array}\right] \cdot \left(\left[\begin{array}{c} \mathbf{V_i^{(n)}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V_i^{(n)}} \end{array}\right][m]\right) \tag{5.200}$$

for all n:

$$\begin{aligned} \left[\begin{array}{c} \mathbf{V_i^{(1)}} \end{array}\right] &= \left[\begin{array}{c} \mathbf{V_i^{(1)}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(1)}} \end{array}\right] \cdot \left(\left[\begin{array}{c} \mathbf{V_i^{(1)}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V_i^{(1)}} \end{array}\right][m]\right) \\ \left[\begin{array}{c} \mathbf{V_i^{(2)}} \end{array}\right] &= \left[\begin{array}{c} \mathbf{V_i^{(2)}} \end{array}\right] - \left[\begin{array}{c} \mathbf{cx_{B''}^{(2)}} \end{array}\right] \cdot \left(\left[\begin{array}{c} \mathbf{V_i^{(2)}} \end{array}\right][\ell] - \left[\begin{array}{c} \mathbf{V_i^{(2)}} \end{array}\right][m]\right) \\ &\vdots \end{aligned} \tag{5.201}$$

9. Free $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B''}^{(1)}} & \mathbf{cx_{B''}^{(2)}} & \cdots & \mathbf{cx_{B''}^{(N_{lines})}} \end{array}\right]$.

10. Allocate and load $\left[\begin{array}{c|c|c|c} \mathbf{V_{i-1}^{(1)}} & \mathbf{V_{i-1}^{(2)}} & \cdots & \mathbf{V_{i-1}^{(N_{lines})}} \end{array}\right]$. The total block is $N_{buses} \times N_{lines}$.

11. Using N_{lines} threads, compute simultaneously

$$\left[\begin{array}{c} \mathbf{V_i^{(n)}} \end{array} \right] = \left[\begin{array}{c} \mathbf{V_i^{(n)}} \end{array} \right] + \left[\begin{array}{c} \mathbf{V_{i-1}^{(n)}} \end{array} \right] \tag{5.202}$$

for all n:

$$\begin{aligned} \left[\begin{array}{c|c|c|c} \mathbf{V_i^{(1)}} & \mathbf{V_i^{(2)}} & \cdots & \mathbf{V_i^{(N_{lines})}} \end{array} \right] = & \left[\begin{array}{c|c|c|c} \mathbf{V_i^{(1)}} & \mathbf{V_i^{(2)}} & \cdots & \mathbf{V_i^{(N_{lines})}} \end{array} \right] \\ & + \left[\begin{array}{c|c|c|c} \mathbf{V_{i-1}^{(1)}} & \mathbf{V_{i-1}^{(2)}} & \cdots & \mathbf{V_{i-1}^{(N_{lines})}} \end{array} \right] \end{aligned} \tag{5.203}$$

12. Free $\left[\begin{array}{c|c|c|c} \mathbf{V_{i-1}^{(1)}} & \mathbf{V_{i-1}^{(2)}} & \cdots & \mathbf{V_{i-1}^{(N_{lines})}} \end{array} \right]$.
13. Write out

$$\left[\begin{array}{c|c|c|c} \mathbf{V_i^{(1)}} & \mathbf{V_i^{(2)}} & \cdots & \mathbf{V_i^{(N_{lines})}} \end{array} \right] \tag{5.204}$$

to computer system main memory for use in the next iteration, and retain it in GPGPU memory for use in calculating the updated bus voltages.

The V Equation, Stott and Alsac Approximation

The equations for calculating the updated bus voltage magnitudes are as follows:

$$\left[\ \mathbf{\Delta V_1^{(n)}}\ \right] = \left[\ \mathbf{\Delta V^{(0)}}\ \right] - c_{B''}^{(n)} \left[\ \mathbf{x_{B''}^{(n)}}\ \right] \cdot \left(\left[\ \mathbf{\Delta V^{(0)}}\ \right][\ell] - \left[\ \mathbf{\Delta V^{(0)}}\ \right][m] \right) \tag{5.205}$$

where the above has been pre-computed, and

$$\left[\ \mathbf{V_1^{(n)}}\ \right] = \left[\ \mathbf{V_0^{(n)}}\ \right] + \left[\ \mathbf{\Delta V_1^{(n)}}\ \right] \tag{5.206}$$

The end goal of this routine is a complete set of vectors

$$\left[\ \mathbf{V_i^{(1)}} \ \middle|\ \mathbf{V_i^{(2)}} \ \middle|\ \cdots \ \middle|\ \mathbf{V_i^{(N_{lines})}}\ \right] \tag{5.207}$$

in GPGPU memory for use in calculating the updated bus voltages and written out to computer system memory and storage for use in the next iteration.

The V Equation, Procedure, Stott and Alsac Approximation

1. $\left[\ \mathbf{\Delta Q^{(1)}} \ \middle|\ \mathbf{\Delta Q^{(2)}} \ \middle|\ \cdots\ \right]$ is still in GPGPU memory.
2. Rename $\left[\ \mathbf{\Delta Q^{(1)}} \ \middle|\ \mathbf{\Delta Q^{(2)}} \ \middle|\ \cdots\ \right]$ to

$$\left[\ \mathbf{V_i^{(1)}} \ \middle|\ \mathbf{V_i^{(2)}} \ \middle|\ \cdots \ \middle|\ \mathbf{V_i^{(N_{lines})}}\ \right] \tag{5.208}$$

and allocate and load the

$$\left[\ \mathbf{\Delta V^{(1)}}\ \middle|\ \mathbf{\Delta V^{(2)}}\ \middle|\ \dots\ \middle|\ \mathbf{\Delta V^{(N_{lines})}}\ \right] \tag{5.209}$$

set of N_{lines} pre-computed $N_{buses} \times 1$ vectors to that location.

3. Allocate and load $\left[\ \mathbf{V^{(1)}_{i-1}}\ \middle|\ \mathbf{V^{(2)}_{i-1}}\ \middle|\ \dots\ \middle|\ \mathbf{V^{(N_{lines})}_{i-1}}\ \right]$. The total block is $N_{buses} \times N_{lines}$.

4. Using N_{lines} threads, compute simultaneously

$$\left[\ \mathbf{V^{(n)}_{i}}\ \right] = \left[\ \mathbf{V^{(n)}_{i}}\ \right] + \left[\ \mathbf{V^{(n)}_{i-1}}\ \right] \tag{5.210}$$

for all n:

$$\begin{aligned}\left[\ \mathbf{V^{(1)}_{i}}\ \middle|\ \mathbf{V^{(2)}_{i}}\ \middle|\ \dots\ \middle|\ \mathbf{V^{(N_{lines})}_{i}}\ \right] &= \left[\ \mathbf{V^{(1)}_{i}}\ \middle|\ \mathbf{V^{(2)}_{i}}\ \middle|\ \dots\ \middle|\ \mathbf{V^{(N_{lines})}_{i}}\ \right] \\ &+ \left[\ \mathbf{V^{(1)}_{i-1}}\ \middle|\ \mathbf{V^{(2)}_{i-1}}\ \middle|\ \dots\ \middle|\ \mathbf{V^{(N_{lines})}_{i-1}}\ \right]\end{aligned} \tag{5.211}$$

5. Free $\left[\ \mathbf{V^{(1)}_{i-1}}\ \middle|\ \mathbf{V^{(2)}_{i-1}}\ \middle|\ \dots\ \middle|\ \mathbf{V^{(N_{lines})}_{i-1}}\ \right]$.

6. Write out

$$\left[\ \mathbf{V^{(1)}_{i}}\ \middle|\ \mathbf{V^{(2)}_{i}}\ \middle|\ \dots\ \middle|\ \mathbf{V^{(N_{lines})}_{i}}\ \right] \tag{5.212}$$

to computer system main memory for use in the next iteration, and retain it in GPGPU memory for use in calculating the updated bus voltages.

Updating the Bus Voltages.

The end goal of this routine is the set of vectors

$$\left[\mathbf{V}', \mathbf{V}''^{(\mathbf{1})} \middle| \mathbf{V}', \mathbf{V}''^{(\mathbf{2})} \middle| \mathbf{V}', \mathbf{V}''^{(\mathbf{3})} \middle| \ldots \right] \tag{5.213}$$

in GPGPU memory for the current update calculations.

Updating the Bus Voltages, Procedure.

1. $\left[\mathbf{V_i^{(1)}} \middle| \mathbf{V_i^{(2)}} \middle| \ldots \middle| \mathbf{V_i^{(N_{lines})}} \right]$ is already in GPGPU memory.
2. Allocate and load $\left[\theta_\mathbf{i}^{(\mathbf{1})} \middle| \theta_\mathbf{i}^{(\mathbf{2})} \middle| \ldots \middle| \theta_\mathbf{i}^{(\mathbf{N_{lines}})} \right]$. The total block is $N_{buses} \times N_{lines}$.
3. Allocate the set of `float2` vectors

$$\left[\mathbf{V}', \mathbf{V}''^{(\mathbf{1})} \middle| \mathbf{V}', \mathbf{V}''^{(\mathbf{2})} \middle| \mathbf{V}', \mathbf{V}''^{(\mathbf{3})} \middle| \ldots \right] \tag{5.214}$$

to fast memory on the GPGPU.

4. Using one thread per $\left[\ \mathbf{V}', \mathbf{V_i''^{(n)}}\ \right]$ vector, compute iteratively for each ℓ:

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \right][\ell].V' &= \left[\ \mathbf{V_i^{(1)}}\ \right][\ell] \\
\theta^{(\ell,1)} &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \right][\ell].V' * \left(\theta^{(\ell,1)} - \frac{\theta^{(\ell,1)} * \theta^{(\ell,1)} * \theta^{(\ell,1)}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V_i''^{(1)}}\ \right][\ell].V' - 0.5 * \theta^{(\ell,1)} * \theta^{(\ell,1)}
\end{aligned} \tag{5.215}$$

simultaneously with:

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \right][\ell].V' &= \left[\ \mathbf{V_i^{(2)}}\ \right][\ell] \\
\theta^{(\ell,2)} &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \right][\ell].V' * \left(\theta^{(\ell,2)} - \frac{\theta^{(\ell,2)} * \theta^{(\ell,2)} * \theta^{(\ell,2)}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V_i''^{(2)}}\ \right][\ell].V' - 0.5 * \theta^{(\ell,2)} * \theta^{(\ell,2)} \\
&\vdots
\end{aligned} \tag{5.216}$$

5. Free the memory locations for the vectors $\left[\ \mathbf{V_i^{(1)}} \middle| \mathbf{V_i^{(2)}} \middle| \cdots\ \right]$ and $\left[\ \mathbf{V_i^{(j)}} \middle| \mathbf{V_i^{(k)}} \middle| \cdots\ \right]$.

6. Free the memory locations for the vectors
$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array}\right]$.

7. The variables $\theta^{(\ell,n)}$ should be thread-local register variables that are freed when their threads complete.

8. Retain $\left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})} \end{array}\right]$ in GPGPU memory for use in computing the current equations.

5.3.4 The $P - \theta$ Iteration: Later Iterations

The initial bus voltages for each of the set of (N-1) contingencies are equal to the bus voltages for the (N-0) base case during the first iteration, but they are distinct for each contingency for later iterations. As a result, the later iterations are more data-intensive than the first iteration.

Calculating the Bus Currents

The set of `float2` vectors

$$\left[\begin{array}{c|c|c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})} & \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} & \cdots \end{array}\right] \tag{5.217}$$

are already in GPGPU memory from the completion of the $Q - V$ iteration.

The end goal of this routine is a complete set of `float2` vectors

$$\left[\begin{array}{c|c|c|c} \mathbf{I}', \mathbf{I}''^{(\mathbf{1})} & \mathbf{I}', \mathbf{I}''^{(\mathbf{2})} & \cdots & \mathbf{I}', \mathbf{I}''^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.218}$$

and a complete set of `float2` vectors

$$\left[\begin{array}{c|c|c|c} \mathbf{V}', \mathbf{V}''^{(1)} & \mathbf{V}', \mathbf{V}''^{(2)} & \ldots & \mathbf{V}', \mathbf{V}''^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.219}$$

in GPGPU memory for use in computing the ΔP equation.

The procedure is that detailed in Section 5.1.3.

Calculating the Bus Currents, Procedure

1. The set of `float2` vectors $\left[\begin{array}{c|c|c} \mathbf{V}', \mathbf{V_i}''^{(1)} & \mathbf{V}', \mathbf{V_i}''^{(2)} & \ldots \end{array}\right]$ are already in GPGPU memory from the completion of the $Q - V$ iteration.

2. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right]$ and $\left[\begin{array}{c} \mathbf{B_{bus}} \end{array}\right]$

3. Allocate and load $\left[\begin{array}{c} \mathbf{G_{bus}} \end{array}\right]$ to the GPGPU according to the sparse-matrix-dense-block-multiplication routine chosen.

4. Allocate the set of `float2` vectors $\left[\begin{array}{c|c|c} \mathbf{I}', \mathbf{I}''^{(1)} & \mathbf{I}', \mathbf{I}''^{(2)} & \ldots \end{array}\right]$ to fast memory on the GPGPU. The total block is $N_{buses} \times N_{lines}$.

5. Using the sparse-matrix-dense-block-multiplication routine chosen, simultaneously compute:

$$\left[\ \mathbf{I}', \mathbf{I}''^{(1)} \ \middle|\ \mathbf{I}', \mathbf{I}''^{(2)} \ \middle|\ \ldots\ \right] = \left[\ \mathbf{G_{bus}}\ \right] \tag{5.220}$$

$$\cdot \left[\ \mathbf{V}', \mathbf{V}''^{(1)} \ \middle|\ \mathbf{V}', \mathbf{V}''^{(2)} \ \middle|\ \ldots\ \right] \tag{5.221}$$

which can also be written as

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(1)}\ \right].I' &= \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(1)}\ \right].V' \\
\left[\ \mathbf{I}', \mathbf{I}''^{(1)}\ \right].I'' &= \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(1)}\ \right].V'' \\
\left[\ \mathbf{I}', \mathbf{I}''^{(2)}\ \right].I' &= \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(2)}\ \right].V' \\
\left[\ \mathbf{I}', \mathbf{I}''^{(2)}\ \right].I'' &= \left[\ \mathbf{G_{bus}}\ \right] \cdot \left[\ \mathbf{V}', \mathbf{V}''^{(2)}\ \right].V'' \\
&\vdots
\end{aligned} \tag{5.222}$$

6. Free $\left[\ \mathbf{G_{bus}}\ \right]$.

7. Allocate and load $\left[\ \mathbf{B_{bus}}\ \right]$ to the GPGPU according to the sparse-matrix-dense-block-multiplication routine chosen.

8. Allocate and load the $N_{lines} \times 1$ vector $\left[\ \mathbf{G_{line}}\ \right]$.

9. Allocate and load the $N_{lines} \times 1$ vector $\left[\ \mathbf{B_{line}}\ \right]$.

10. Using the sparse-matrix-dense-block-multiplication routine chosen, simultaneously compute:

$$\begin{aligned}
\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I' - \left[\ \mathbf{B_{bus}}\ \right]\cdot\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \right].V'' \\
\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I'' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{1})}\ \right].I'' + \left[\ \mathbf{B_{bus}}\ \right]\cdot\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{1})}\ \right].V' \\
\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I' - \left[\ \mathbf{B_{bus}}\ \right]\cdot\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \right].V'' \\
\left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I'' &= \left[\ \mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\ \right].I'' + \left[\ \mathbf{B_{bus}}\ \right]\cdot\left[\ \mathbf{V}',\mathbf{V}''^{(\mathbf{2})}\ \right].V' \\
&\vdots
\end{aligned} \tag{5.223}$$

11. Free $\left[\ \mathbf{B_{bus}}\ \right]$.

12. Allocate an $N_{lines} \times 1$ vector of CUDA aligned type `float2`, $\left[\ \mathbf{I'_{adj}},\mathbf{I''_{adj}}\ \right]$.

13. Using N_{lines} threads, for each single contingency n in which the line from bus ℓ to bus m is out of service, simultaneously compute:

$$\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I'_{adj} &= \left[\ \mathbf{G_{line}}\ \right][n] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V'\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I''_{adj} &= \left[\ \mathbf{G_{line}}\ \right][n] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V''\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I'_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I'_{adj} - \left[\ \mathbf{B_{line}}\ \right][n] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V''\right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I''_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][n].I''_{adj} + \left[\ \mathbf{B_{line}}\ \right][n] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][\ell].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(n)}}\ \right][m].V'\right)
\end{aligned} \tag{5.224}$$

for all n:

$$
\begin{aligned}
\left[\begin{array}{c} \mathbf{I'_{adj}}, \mathbf{I''_{adj}} \end{array}\right][1].I'_{adj} = & \left[\begin{array}{c} \mathbf{G_{line}} \end{array}\right][1] \\
& \cdot \left(\left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][\ell_1].V' - \left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][m_1].V'\right) \\
\left[\begin{array}{c} \mathbf{I'_{adj}}, \mathbf{I''_{adj}} \end{array}\right][1].I''_{adj} = & \left[\begin{array}{c} \mathbf{G_{line}} \end{array}\right][1] \\
& \cdot \left(\left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][\ell_1].V'' - \left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][m_1].V''\right) \\
\left[\begin{array}{c} \mathbf{I'_{adj}}, \mathbf{I''_{adj}} \end{array}\right][1].I'_{adj} = & \left[\begin{array}{c} \mathbf{I'_{adj}}, \mathbf{I''_{adj}} \end{array}\right][1].I'_{adj} - \left[\begin{array}{c} \mathbf{B_{line}} \end{array}\right][1] \\
& \cdot \left(\left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][\ell_1].V'' - \left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][m_1].V''\right) \\
\left[\begin{array}{c} \mathbf{I'_{adj}}, \mathbf{I''_{adj}} \end{array}\right][1].I''_{adj} = & \left[\begin{array}{c} \mathbf{I'_{adj}}, \mathbf{I''_{adj}} \end{array}\right][1].I''_{adj} + \left[\begin{array}{c} \mathbf{B_{line}} \end{array}\right][1] \\
& \cdot \left(\left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][\ell_1].V' - \left[\begin{array}{c} \mathbf{V'}, \mathbf{V''^{(1)}} \end{array}\right][m_1].V'\right)
\end{aligned}
\tag{5.225}
$$

$$
\begin{aligned}
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I'_{adj} &= \left[\ \mathbf{G_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell_2].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][m_2].V' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I''_{adj} &= \left[\ \mathbf{G_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell_2].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][m_2].V'' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I'_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I'_{adj} - \left[\ \mathbf{B_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell_2].V'' - \left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][m_2].V'' \right) \\
\left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I''_{adj} &= \left[\ \mathbf{I'_{adj}}, \mathbf{I''_{adj}}\ \right][2].I''_{adj} + \left[\ \mathbf{B_{line}}\ \right][2] \\
&\cdot \left(\left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][\ell_2].V' - \left[\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \right][m_2].V' \right) \\
&\vdots
\end{aligned}
\tag{5.226}
$$

14. Free $\left[\ \mathbf{G_{line}}\ \right]$.

15. Free $\left[\ \mathbf{B_{line}}\ \right]$.

16. Using N_{lines} threads, simultaneously compute

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][\ell].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{n})}\ \right][m].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][n].I''_{adj}
\end{aligned} \tag{5.227}$$

for all n:

$$\begin{aligned}
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][\ell_1].I'' + \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I''_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I'_{adj} \\
\left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I'' &= \left[\ \mathbf{I}', \mathbf{I}''^{(\mathbf{1})}\ \right][m_1].I'' - \left[\ \mathbf{I}'_{\mathbf{adj}}, \mathbf{I}''_{\mathbf{adj}}\ \right][1].I''_{adj}
\end{aligned} \tag{5.228}$$

$$\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I' = \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I' + \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][2].I'_{adj} \quad (5.229)$$

$$\begin{aligned}
\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I'' &= \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][\ell_2].I'' + \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][2].I''_{adj} \\
\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I' &= \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I' - \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][2].I'_{adj} \\
\left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I'' &= \left[\begin{array}{c}\mathbf{I}',\mathbf{I}''^{(\mathbf{2})}\end{array}\right][m_2].I'' - \left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right][2].I''_{adj} \\
&\vdots
\end{aligned} \quad (5.230)$$

17. Free $\left[\begin{array}{c}\mathbf{I}'_{\mathbf{adj}},\mathbf{I}''_{\mathbf{adj}}\end{array}\right]$.

18. Retain $\left[\begin{array}{c|c|c|c}\mathbf{I}',\mathbf{I}''^{(\mathbf{1})} & \mathbf{I}',\mathbf{I}''^{(\mathbf{2})} & \dots & \mathbf{I}',\mathbf{I}''^{(\mathbf{N_{lines}})}\end{array}\right]$ in GPGPU memory for use in computing the ΔP equation.

19. Retain $\left[\begin{array}{c|c|c|c}\mathbf{V}',\mathbf{V}''^{(\mathbf{1})} & \mathbf{V}',\mathbf{V}''^{(\mathbf{2})} & \dots & \mathbf{V}',\mathbf{V}''^{(\mathbf{N_{lines}})}\end{array}\right]$ in GPGPU memory for use in computing the ΔP equation.

The ΔP Equation, Later Iterations

Where the calculation for later iterations differs from the first iteration is that the bus voltages are distinct for the various (N-1) contingencies.

The end goals of this routine are:

1. The vector $\left[\begin{array}{c}\mathbf{\Delta P_{max,i}}\end{array}\right]$ written out to computer system main memory to use to test for convergence.

2. A complete set of vectors

$$\left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \cdots\ \right] \tag{5.231}$$

in GPGPU memory for use in calculating the θ equation.

The ΔP Equation, Later Iterations, Procedure

1. $\left[\ \mathbf{I'}, \mathbf{I''^{(1)}}\ \middle|\ \mathbf{I'}, \mathbf{I''^{(2)}}\ \middle|\ \cdots\ \middle|\ \mathbf{I'}, \mathbf{I''^{(N_{lines})}}\ \right]$ is still in GPGPU memory.
2. $\left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \middle|\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \middle|\ \cdots\ \middle|\ \mathbf{V'}, \mathbf{V''^{(N_{lines})}}\ \right]$ is still in GPGPU memory.
3. Use N_{lines} threads to compute simultaneously :

$$\begin{aligned}\left[\ \mathbf{I'}, \mathbf{I''^{(1)}}\ \middle|\ \mathbf{I'}, \mathbf{I''^{(2)}}\ \middle|\ \cdots\ \right] = &\left[\ \mathbf{I'}, \mathbf{I''^{(1)}}\ \middle|\ \mathbf{I'}, \mathbf{I''^{(2)}}\ \middle|\ \cdots\ \right] \\ &* . \left[\ \mathbf{V'}, \mathbf{V''^{(1)}}\ \middle|\ \mathbf{V'}, \mathbf{V''^{(2)}}\ \middle|\ \cdots\ \right]\end{aligned} \tag{5.232}$$

which can also be written:

$$\begin{gathered}
\left[\ \mathbf{I}',\mathbf{I}''^{(1)}\ \right].I' = \left[\ \mathbf{I}',\mathbf{I}''^{(1)}\ \right].I' * .\left[\ \mathbf{V}',\mathbf{V}''^{(1)}\ \right].V' \\
\left[\ \mathbf{I}',\mathbf{I}''^{(1)}\ \right].I'' = \left[\ \mathbf{I}',\mathbf{I}''^{(1)}\ \right].I'' * .\left[\ \mathbf{V}',\mathbf{V}''^{(1)}\ \right].V'' \\
\left[\ \mathbf{I}',\mathbf{I}''^{(2)}\ \right].I' = \left[\ \mathbf{I}',\mathbf{I}''^{(2)}\ \right].I' * .\left[\ \mathbf{V}',\mathbf{V}''^{(2)}\ \right].V' \\
\left[\ \mathbf{I}',\mathbf{I}''^{(2)}\ \right].I'' = \left[\ \mathbf{I}',\mathbf{I}''^{(2)}\ \right].I'' * .\left[\ \mathbf{V}',\mathbf{V}''^{(2)}\ \right].V'' \\
\vdots
\end{gathered} \tag{5.233}$$

4. Free $\left[\ \mathbf{V}',\mathbf{V}''^{(1)}\ \middle|\ \mathbf{V}',\mathbf{V}''^{(2)}\ \middle|\ \ldots\ \middle|\ \mathbf{V}',\mathbf{V}''^{(\mathbf{N_{lines}})}\ \right]$.

5. Allocate and load the $N_{buses} \times 1$ scalar vector $\left[\ \mathbf{P_{sched}}\ \right]$.

6. Allocate $\left[\ \boldsymbol{\Delta}\mathbf{P}^{(1)}\ \middle|\ \boldsymbol{\Delta}\mathbf{P}^{(2)}\ \middle|\ \ldots\ \middle|\ \boldsymbol{\Delta}\mathbf{P}^{(\mathbf{N_{lines}})}\ \right]$. The entire block is $N_{buses} \times N_{lines}$.

7. Use N_{lines} threads and the CUDA broadcast functionality to initialize

$$\left[\ \boldsymbol{\Delta}\mathbf{P}^{(\mathbf{n})}\ \right] = \left[\ \mathbf{P_{sched}}\ \right] \tag{5.234}$$

for all n:

$$\left[\ \boldsymbol{\Delta}\mathbf{P}^{(1)}\ \middle|\ \boldsymbol{\Delta}\mathbf{P}^{(2)}\ \middle|\ \ldots\ \middle|\ \boldsymbol{\Delta}\mathbf{P}^{(\mathbf{N_{lines}})}\ \right] = \left[\ \mathbf{P_{sched}}\ \right] \tag{5.235}$$

8. Free $\left[\mathbf{P_{sched}} \right]$.

9. Use N_{lines} threads to compute simultaneously

$$\begin{aligned} \left[\mathbf{\Delta P^{(n)}} \right] &= \left[\mathbf{\Delta P^{(n)}} \right] - \left[\mathbf{I', I''^{(n)}} \right] .I' \\ \left[\mathbf{\Delta P^{(n)}} \right] &= \left[\mathbf{\Delta P^{(n)}} \right] - \left[\mathbf{I', I''^{(n)}} \right] .I'' \end{aligned} \tag{5.236}$$

for all n:

$$\begin{aligned} \left[\mathbf{\Delta P^{(1)}} \right] &= \left[\mathbf{\Delta P^{(1)}} \right] - \left[\mathbf{I', I''^{(1)}} \right] .I' \\ \left[\mathbf{\Delta P^{(1)}} \right] &= \left[\mathbf{\Delta P^{(1)}} \right] - \left[\mathbf{I', I''^{(1)}} \right] .I'' \end{aligned} \tag{5.237}$$

$$\begin{aligned} \left[\mathbf{\Delta P^{(2)}} \right] &= \left[\mathbf{\Delta P^{(2)}} \right] - \left[\mathbf{I', I''^{(2)}} \right] .I' \\ \left[\mathbf{\Delta P^{(2)}} \right] &= \left[\mathbf{\Delta P^{(2)}} \right] - \left[\mathbf{I', I''^{(2)}} \right] .I'' \\ &\vdots \end{aligned} \tag{5.238}$$

10. Free $\left[\mathbf{I', I''^{(1)}} \middle| \mathbf{I', I''^{(2)}} \middle| \cdots \middle| \mathbf{I', I''^{(N_{lines})}} \right]$.

11. Allocate the $N_{lines} \times 1$ scalar vector $\left[\mathbf{\Delta P_{max,i}} \right]$.

12. Use N_{lines} threads to compute simultaneously

$$\Delta P_{max,i}[n] = \max\left(\left[\begin{array}{c} \mathbf{\Delta P^{(n)}} \end{array}\right]\right) \tag{5.239}$$

for all n:

$$\begin{aligned} \Delta P_{max,i}[1] &= \max\left(\left[\begin{array}{c} \mathbf{\Delta P^{(1)}} \end{array}\right]\right) \\ \Delta P_{max,i}[2] &= \max\left(\left[\begin{array}{c} \mathbf{\Delta P^{(2)}} \end{array}\right]\right) \\ &\vdots \end{aligned} \tag{5.240}$$

13. Write out $\left[\begin{array}{c} \mathbf{\Delta P_{max,i}} \end{array}\right]$ to the computer system main memory and storage to use to test for convergence.

14. Free $\left[\begin{array}{c} \mathbf{\Delta P_{max,i}} \end{array}\right]$.

15. Allocate and load $\left[\begin{array}{c|c|c|c} \mathbf{V^{(1)}} & \mathbf{V^{(2)}} & \ldots & \mathbf{V^{(N_{lines})}} \end{array}\right]$. The entire block is $N_{buses} \times N_{lines}$.

16. Use N_{lines} threads to compute simultaneously:

$$\left[\begin{array}{c} \mathbf{\Delta P^{(n)}} \end{array}\right] = \left[\begin{array}{c} \mathbf{\Delta P^{(n)}} \end{array}\right] ./ \left[\begin{array}{c} \mathbf{V^{(n)}} \end{array}\right] \tag{5.241}$$

for all n:

$$\left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \ldots\ \right] = \left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \ldots\ \right] ./ \left[\ \mathbf{V^{(1)}}\ \middle|\ \mathbf{V^{(2)}}\ \middle|\ \ldots\ \right] \tag{5.242}$$

17. Free $\left[\ \mathbf{V^{(1)}}\ \middle|\ \mathbf{V^{(2)}}\ \middle|\ \ldots\ \middle|\ \mathbf{V^{(N_{lines})}}\ \right]$.

18. Retain $\left[\ \mathbf{\Delta P^{(1)}}\ \middle|\ \mathbf{\Delta P^{(2)}}\ \middle|\ \ldots\ \right]$ in GPGPU memory for use in calculating the θ equation.

The θ Equation, Later Iterations, Full $\Delta\theta$

Where the calculation for later iterations differs from the first iteration is that the bus voltages are distinct for the various (N-1) contingencies.

The end goal of this routine is a complete set of vectors

$$\left[\ \theta_{\mathbf{i}}^{\mathbf{(1)}}\ \middle|\ \theta_{\mathbf{i}}^{\mathbf{(2)}}\ \middle|\ \ldots\ \middle|\ \theta_{\mathbf{i}}^{\mathbf{(N_{lines})}}\ \right] \tag{5.243}$$

in GPGPU memory for use in calculating the updated bus voltages and written out to computer system memory and storage for use in the next iteration.

However, if not all data needed for this routine will fit in GPGPU memory as it is needed, priority should be given to keeping the sparse matrix $\left[\ \mathbf{B'^{(0)}}\ \right]^{-1}$ in GPGPU memory while swapping out sections of blocks of dense vectors, since it is the sparse matrix that provides data needed by all (N-1) contingencies.

The θ Equation, Later Iterations, Procedure, Full $\Delta\theta$

1. $\left[\begin{array}{c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \cdots \end{array} \right]$ is still in GPGPU memory.

2. Choose a sparse-matrix-dense-block-multiplication routine based on the properties of $\left[\mathbf{B'^{(0)}} \right]^{-1}$.

3. Allocate and load $\left[\mathbf{B'^{(0)}} \right]^{-1}$ according to the sparse-matrix-dense-block-multiplication routine chosen.

4. Allocate $\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array} \right]$. The total block is $N_{buses} \times N_{lines}$.

5. Using the sparse-matrix-dense-block-multiplication routine chosen, compute:

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array} \right] = \left[\mathbf{B'^{(0)}} \right]^{-1} \cdot \left[\begin{array}{c|c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \cdots & \mathbf{\Delta P^{(N_{lines})}} \end{array} \right] \tag{5.244}$$

6. Free $\left[\mathbf{B'^{(0)}} \right]^{-1}$.

7. Rename $\left[\begin{array}{c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \cdots \end{array} \right]$ to $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B'}^{(1)}} & \mathbf{cx_{B'}^{(2)}} & \cdots & \mathbf{cx_{B'}^{(N_{lines})}} \end{array} \right]$ and load $\left[\begin{array}{c|c|c|c} \mathbf{cx_{B'}^{(1)}} & \mathbf{cx_{B'}^{(2)}} & \cdots & \mathbf{cx_{B'}^{(N_{lines})}} \end{array} \right]$ to that location on the GPGPU. The total block is $N_{buses} \times N_{lines}$.

8. Using N_{lines} threads, compute simultaneously:

$$\left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right] = \left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right] - \left[\ \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{n})}\ \right] \cdot \left(\left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right][\ell] - \left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right][m] \right) \tag{5.245}$$

for all n:

$$\begin{aligned} \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right] &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right] - \left[\ \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{1})}\ \right] \cdot \left(\left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right][\ell] - \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right][m] \right) \\ \left[\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \right] &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \right] - \left[\ \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{2})}\ \right] \cdot \left(\left[\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \right][\ell] - \left[\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \right][m] \right) \\ &\vdots \end{aligned} \tag{5.246}$$

9. Free $\left[\ \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{1})}\ \middle|\ \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \mathbf{cx}_{\mathbf{B'}}^{(\mathbf{N_{lines}})}\ \right]$.
10. Allocate and load $\left[\ \theta_{\mathbf{i-1}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i-1}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i-1}}^{(\mathbf{N_{lines}})}\ \right]$. The total block is $N_{buses} \times N_{lines}$.
11. Using N_{lines} threads, compute simultaneously

$$\left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right] = \left[\ \theta_{\mathbf{i}}^{(\mathbf{n})}\ \right] + \left[\ \theta_{\mathbf{i-1}}^{(\mathbf{n})}\ \right] \tag{5.247}$$

for all n:

$$\left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})}\ \right] = \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})}\ \right] + \left[\ \theta_{\mathbf{i-1}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i-1}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i-1}}^{(\mathbf{N_{lines}})}\ \right] \tag{5.248}$$

12. Free $\left[\ \theta_{\mathbf{i-1}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i-1}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i-1}}^{(\mathbf{N_{lines}})}\ \right]$.

13. Write out

$$\left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})}\ \right] \tag{5.249}$$

to computer system main memory for use in the next iteration, and retain it in GPGPU memory for use in calculating the updated bus voltages.

The θ Equation, Stott and Alsac Approximation

Where the calculation for later iterations differs from the first iteration is that the bus voltages are distinct for the various (N-1) contingencies.

The end goal of this routine is a complete set of vectors

$$\left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \cdots\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})}\ \right] \tag{5.250}$$

in GPGPU memory for use in calculating the updated bus voltages and written out to computer system memory and storage for use in the next iteration.

The θ Equation, Procedure, Stott and Alsac Approximation

1. $\left[\begin{array}{c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \cdots \end{array} \right]$ is still in GPGPU memory.

2. Rename $\left[\begin{array}{c|c|c} \mathbf{\Delta P^{(1)}} & \mathbf{\Delta P^{(2)}} & \cdots \end{array} \right]$ to

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array} \right] \tag{5.251}$$

and allocate and load the

$$\left[\begin{array}{c|c|c|c} \mathbf{\Delta}\theta^{(\mathbf{1})} & \mathbf{\Delta}\theta^{(\mathbf{2})} & \cdots & \mathbf{\Delta}\theta^{(\mathbf{N_{lines}})} \end{array} \right] \tag{5.252}$$

set of N_{lines} pre-computed $N_{buses} \times 1$ vectors to that location.

3. Allocate and load $\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i-1}}^{(\mathbf{1})} & \theta_{\mathbf{i-1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i-1}}^{(\mathbf{N_{lines}})} \end{array} \right]$. The total block is $N_{buses} \times N_{lines}$.

4. Using N_{lines} threads, compute simultaneously

$$\left[\theta_{\mathbf{i}}^{(\mathbf{n})} \right] = \left[\theta_{\mathbf{i}}^{(\mathbf{n})} \right] + \left[\theta_{\mathbf{i-1}}^{(\mathbf{n})} \right] \tag{5.253}$$

for all n:

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array}\right] = \left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array}\right] + \left[\begin{array}{c|c|c|c} \theta_{\mathbf{i-1}}^{(\mathbf{1})} & \theta_{\mathbf{i-1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i-1}}^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.254}$$

5. Free $\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i-1}}^{(\mathbf{1})} & \theta_{\mathbf{i-1}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i-1}}^{(\mathbf{N_{lines}})} \end{array}\right]$.

6. Write out

$$\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array}\right] \tag{5.255}$$

to computer system main memory for use in the next iteration, and retain it in GPGPU memory for use in calculating the updated bus voltages.

Updating the Bus Voltages.

The end goal of this routine is the set of vectors

$$\left[\begin{array}{c|c|c|c} \mathbf{V}', \mathbf{V}''^{(\mathbf{1})} & \mathbf{V}', \mathbf{V}''^{(\mathbf{2})} & \mathbf{V}', \mathbf{V}''^{(\mathbf{3})} & \cdots \end{array}\right] \tag{5.256}$$

in GPGPU memory for the current update calculations.

Updating the Bus Voltages, Procedure.

1. $\left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \ldots\ \middle|\ \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})}\ \right]$ is already in GPGPU memory.

2. Allocate and load $\left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{1})}\ \middle|\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{2})}\ \middle|\ \ldots\ \middle|\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{N_{lines}})}\ \right]$. The total block is $N_{buses} \times N_{lines}$.

3. Allocate the set of `float2` vectors

$$\left[\ \mathbf{V}', \mathbf{V}''^{(\mathbf{1})}\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{2})}\ \middle|\ \mathbf{V}', \mathbf{V}''^{(\mathbf{3})}\ \middle|\ \ldots\ \right] \tag{5.257}$$

   to fast memory on the GPGPU.

4. Using one thread per $\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})}\ \right]$ vector, compute iteratively for each ℓ:

$$\begin{aligned}
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' &= \left[\ \mathbf{V}_{\mathbf{i}}^{(\mathbf{1})}\ \right][\ell] \\
\theta^{(\ell,1)} &= \left[\ \theta_{\mathbf{i}}^{(\mathbf{1})}\ \right][\ell] \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V'' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' * \left(\theta^{(\ell,1)} - \frac{\theta^{(\ell,1)} * \theta^{(\ell,1)} * \theta^{(\ell,1)}}{6}\right) \\
\left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' &= \left[\ \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{1})}\ \right][\ell].V' - 0.5 * \theta^{(\ell,1)} * \theta^{(\ell,1)}
\end{aligned} \tag{5.258}$$

   simultaneously with:

$$\begin{aligned}
\left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} \end{array}\right][\ell].V' &= \left[\begin{array}{c} \mathbf{V}_{\mathbf{i}}^{(\mathbf{2})} \end{array}\right][\ell] \\
\theta^{(\ell,2)} &= \left[\begin{array}{c} \theta_{\mathbf{i}}^{(\mathbf{2})} \end{array}\right][\ell] \\
\left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} \end{array}\right][\ell].V'' &= \left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} \end{array}\right][\ell].V' * \left(\theta^{(\ell,2)} - \frac{\theta^{(\ell,2)} * \theta^{(\ell,2)} * \theta^{(\ell,2)}}{6}\right) \\
\left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} \end{array}\right][\ell].V' &= \left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{2})} \end{array}\right][\ell].V' - 0.5 * \theta^{(\ell,2)} * \theta^{(\ell,2)} \\
&\vdots
\end{aligned} \tag{5.259}$$

5. Free the memory locations for the vectors $\left[\begin{array}{c|c|c} \mathbf{V}_{\mathbf{i}}^{(\mathbf{1})} & \mathbf{V}_{\mathbf{i}}^{(\mathbf{2})} & \cdots \end{array}\right]$ and $\left[\begin{array}{c|c|c} \mathbf{V}_{\mathbf{i}}^{(\mathbf{j})} & \mathbf{V}_{\mathbf{i}}^{(\mathbf{k})} & \cdots \end{array}\right]$.
6. Free the memory locations for the vectors $\left[\begin{array}{c|c|c|c} \theta_{\mathbf{i}}^{(\mathbf{1})} & \theta_{\mathbf{i}}^{(\mathbf{2})} & \cdots & \theta_{\mathbf{i}}^{(\mathbf{N_{lines}})} \end{array}\right]$.
7. The variables $\theta^{(\ell,n)}$ should be thread-local register variables that are freed when their threads complete.
8. Retain $\left[\begin{array}{c} \mathbf{V}', \mathbf{V}_{\mathbf{i}}''^{(\mathbf{n})} \end{array}\right]$ in GPGPU memory for use in computing the current equations.

5.4 Computing Contingencies for (N-x)

5.4.1 Computing Contingencies for (N-1-1)

Computing the full set of (N-1-1) contingencies (which number $N(N-1)$ contingencies in total as discussed in Chapter 2) inherently involves computing the full set of (N-1)

contingencies first and using each of those solutions as the base case for the set of contingencies found by removing each remaining individual line in succession from each of the (N-1) base cases. For each of the new set of contingencies computed from a single (N-1) base case, the bus voltage magnitude and angles from the solution of the (N-1) base case serve as inputs for each of the contingencies computed from that base case.

While creating a new $\left[\ \mathbf{G_{bus}^{(n)}}\ \right]$ and $\left[\ \mathbf{B_{bus}^{(n)}}\ \right]$ for each of the (N-1) base cases is an option, the bus currents could still be computed using the $\left[\ \mathbf{G_{bus}^{(0)}}\ \right]$ and $\left[\ \mathbf{B_{bus}^{(0)}}\ \right]$ from the original (N-0) base case and applying the current correction methodology and equations from Section 4.4 to two lines at a time instead of one line at a time.

The line outage bus current corrections double in the amount of computation required per contingency when the number of outages doubles, so the relatively minor work of updating each $\left[\ \mathbf{G_{bus}^{(n)}}\ \right]$ and $\left[\ \mathbf{B_{bus}^{(n)}}\ \right]$ from each $\left[\ \mathbf{G_{bus}^{(0)}}\ \right]$ and $\left[\ \mathbf{B_{bus}^{(0)}}\ \right]$ seems like the correct choice from the standpoint of merely managing the number of computations. However, the increase in computations (which add up with every iteration) has to be weighed against the increased number of matrices that must be transported from the computer system main memory to the GPGPU: $(N-1)$ different $\left[\ \mathbf{G_{bus}}\ \right]$ matrices instead of one, and $(N-1)$ different $\left[\ \mathbf{B_{bus}}\ \right]$ instead of one. Unless the specific GPGPU and implementation of this method completely mask the latency of transporting these matrices over the system bus, the amount of extra computation is probably faster.

To estimate the multiplicative factor increase in current corrections over the set of (N-1) contingencies, if we count each line outage as one current correction, the number of current corrections is the number of line outages per contingency multiplied the number

	(N-1-1)	(N-2)	(N-3)
Contingencies	$N(N-1)$	$\frac{N(N-1)}{2}$	$\frac{N(N-1)(N-2)}{6}$
Factor Increase in Current Corrections	$2 * \frac{N(N-1)}{N}$	$2 * \frac{N(N-1)}{2N}$	$3 * \frac{N(N-1)(N-2)}{6N}$
=	$2 * (N-1)$	$N-1$	$\frac{(N-1)(N-2)}{2}$
Base Cases	N	N	$\frac{N(N-1)}{2}$
Factor Increase in Transported Matrices	N	N	$\frac{N(N-1)}{2}$

Table 4: Differences in computational and data transport burden versus the set of (N-1) contingencies.

of contingencies. Thus the factor increase in number of current corrections for the (N-1-1) set of contingencies over the set of (N-1) contingencies is two line outages per contingency multiplied by $\frac{N(N-1)(N-2)}{6}$ contingencies , divided by the one line outage per contingency multiplied by N contingencies. This is shown in Table 4.

The multiplicative factor increase in number of matrices that must be transported is the number of (N-1-1) base cases (N) divided by the number of (N-1) base cases (1).

While creating a new $\left[\mathbf{B}'^{(\mathbf{n})}\right]^{-1}$ and $\left[\mathbf{B}''^{(\mathbf{n})}\right]^{-1}$ for each of the (N-1) base cases, the θ and V equations could still be computed using the $\left[\mathbf{B}'^{(\mathbf{0})}\right]^{-1}$ and $\left[\mathbf{B}''^{(\mathbf{0})}\right]^{-1}$ from the original (N-1) base case and using the matrix inversion lemma methodology and equations from Section 4.5.4 applied to two lines at a time instead of one line at a time.

Since the scalar constants $c_{B'}^{(n)}$ and $c_{B'}^{(n)}$ and the vector constants $\left[\mathbf{x}_{\mathbf{B}'}^{(\mathbf{n})}\right]$ and $\left[\mathbf{x}_{\mathbf{B}''}^{(\mathbf{n})}\right]$ are pre-computed, the increase in computation during the iterations of the FDPF solver are

solely a function of the increased entries in $\left[\mathbf{x}_{\mathbf{B'}}^{(\mathbf{n})} \right]$ and $\left[\mathbf{x}_{\mathbf{B''}}^{(\mathbf{n})} \right]$, if there is an increase in their entries. There is an increase in pre-computation, however. Computing and transporting a new set of $N \left[\mathbf{B}'^{(\mathbf{n})} \right]^{-1}$ and $N \left[\mathbf{B}''^{(\mathbf{n})} \right]^{-1}$ is a substantially great burden in both computation and data transport.

When using the Stott and Alsac approximation detailed in Section 4.5.6, the question of computing new matrix inverses and transporting them to the GPGPU does not arise. However, since under this approximation, the values of $\left[\mathbf{\Delta V}^{(\mathbf{n})} \right]$ and $\left[\mathbf{\Delta}\theta^{(\mathbf{n})} \right]$ are entirely decoupled from the iterations of the solver. While Stott and Alsac state this approximation can be applied the set of (N-2) contingencies using the the base case (N-0) data [70], they originally found it faster than inverting a new set of $\left[\mathbf{B}'^{(\mathbf{n})} \right]^{-1}$, and $\left[\mathbf{B}''^{(\mathbf{n})} \right]^{-1}$ matrices for at most three multiple outages [70], presumably because of additional iterations required for convergence under the approximate method. Since the more accurate method presented in this thesis that uses the matrix inversion lemma to avoid calculating new matrix inverses but without the Stott and Alsac approximation represents an intermediate step in computational burden between the Stott and Alsac approximation and computing a new set of matrix inverses for each of the (N-1) base cases, this method may well prove the best choice for (N-1-1), (N-2), and (N-3) contingency analysis. However, the speed of the Stott and Alsac approximation makes it of vital interest for (N-1) contingency analysis that must be performed in a tight timeframe.

5.4.2 Computing Contingencies for (N-2)

For computing the full set of (N-2) contingencies (which number $\frac{N(N-1)}{2}$ contingencies in total), the bus voltage magnitude and angles from the solution of the (N-0) base case serve as inputs for each of the contingencies, since the study of the set of (N-2) contingencies is the study of when two lines are removed from service at once.

Otherwise, the same issues apply as for (N-1-1) with respect to building new and transporting new matrices. The multiplicative factor increase in current corrections over the set of (N-1) contingencies is half the multiplicative factor increase for the set of (N-1-1) contingencies because the number of distinct contingencies is half as many as shown in Table 4. If creating a new set of base case matrices is chosen instead, the multiplicative factor increase in transported matrices over the set of (N-1) contingencies is the same as for (N-1-1) because the number of base cases is the same.

5.4.3 Computing Contingencies for (N-3)

For computing the full set of (N-3) contingencies (which number $\frac{N(N-1)(N-2)}{6}$ contingencies in total) the bus voltage magnitude and angles from the solution of the (N-0) base case serve as inputs for each of the contingencies, since the study of the set of (N-3) contingencies is the study of when three lines are removed from service at once.

The multiplicative factor increase in current corrections over the set of (N-1) contingencies is shown in Table 4, as is the multiplicative factor increase in transported matrices over the set of (N-1) contingencies if creating a new set of base case matrices is chosen instead. As with the previous cases, which option to choose in a GPGPU implementation

depends on whether the specific implementation completely masks the latency of transporting these matrices over the system bus. This continues to be true for (N-x) where $x > 3$.

5.5 Summary

This chapter gave the details of the method proposed in this thesis for accelerating (N-1) contingency analysis using a GPGPU. Common routines that are used repeatedly were discussed first, followed by routines that can be pre-computed outside the iterations of the FDPF solution algorithm. Then the entirety of the proposed method was detailed, with the differences noted where the Stott and Alsac approximation may be applied, followed by a discussion of how smoothly this method may be applied to (N-1-1), (N-2), and (N-3) contingency analysis and some of the associated implications in computational burden and data movement.

Chapter 6

Recent Work in and Early Implementations of GPGPGU Contingency Analysis

Several attempts have been made from 2007 onward to accelerate computation of power flow algorithms or power systems contingency analysis by using a GPGPU as a floating-point accelerator. Of those recorded in the literature, most of them simply chose part or parts of a power flow algorithm to implement on a GPGPU. There is one implementation discussed below (in two papers) of an entire power flow run on a GPGPU without examining the applications to contingency analysis, and two implementations of running multiple power flows on a GPU at the same time. The only implementation below of (N-1) contingency analysis on a GPU is implemented on a pre-CUDA GPU [71], which is a GPU without the general-purpose programming interface. This requires that in writing GPU code arithmetic operations must be translated into pixel manipulations, since the GPU instruction sets that existed at the time only had pixel instructions for any kind of floating-point computation. Some of the other methods below adapted methods developed for pre-CUDA GPUs onto GPGPUs.

6.1 Implementing Parts of a Power Flow on the GPGPU

In [72], the author started from an existing Conjugate Gradient method developed for pre-CUDA GPUs and developed a Biconjugate Gradient method $A^{-1}x = b$ solver for GPGPUs. The author also chose a sparse matrix storage method and adapted an existing matrix-vector multiplication routine for the GPGPU and used that to accelerate the sparse matrix-vector multiplication portions of the power flow code. These were combined with a Newton's Method power flow algorithm to accelerated the power flow using GPGPU computation, though computation of the Jacobian matrix was reserved to the computer system's CPU. The test case was the IEEE 118-bus case. Speedup was 2.1 compared to CPU code.

In [56], the authors developed a Conjugate Gradient method with Jacobi preconditioning $A^{-1}x = b$ solver and applied it to test matrices intended to represent DC power flow constant $\left[\mathbf{B} \right]$ matrices. The test matrices were developed for theoretical 494, 685, 662, and 1138 bus systems, and their entries were then multiplied to artificially extend the matrix size. Solutions of $A^{-1}x = b$ solver had greater speedups on the GPGPU for larger matrices than for smaller matrices, with a speed-up of 49.3 for the 662 bus matrix multiplied 336 times, and a speed-up of 36.4 for the 1138 bus matrix multiplied 336 times.

In [73], the authors used the GPGPU to compute the admittance matrix and to perform Gaussian elimination for the solution of $A^{-1}x = b$ equations, while the remainder of the power flow was reserved to the computer system's CPU. This method was used to compare the effects on three different power flow algorithms: Gauss-Seidel, Newton-Raphson, and the FDPF. The results for the IEEE 9, 30, 118, and 300 bus cases all had marginal speedups,

however the results for the Shandong Province system with 974 buses 1449 lines had speed ups of 0.062 for Gauss-Seidel, 53.656 for Newton-Raphson, and 27.0527 for the FDPF.

In [74], the authors took an existing CPU-based AC Newton-Rapshon Power Flow code and used the GPGPU to accelerate four sections of the code; 1) calculation of the admittance matrix, 2) calculation of the real and reactive power Δs, 3) calculation of the Jacobian, and the 4) the solution of $A^{-1}x = b$ equations. The results were a speedup of 6.1 for the IEEE 14 bus case, 2.6 for the IEEE 30 bus case, and 2.1 for the IEEE 118 bus case.

6.2 Implementing a Power Flow on the GPGPU

In [75], the authors implement Newton-Raphson and Gauss-Jacobi based power flow algorithms on a GPGPU using a proprietary library of sparse matrix routines for GPGPUs called CULAtools. They also detail a GPGPU routine for computing power injections from an updated set of generator output powers. Results for the Newton-Raphson method were a speedup of 4.5 for the IEEE 678 bus case, a speedup of 8.5 for the IEEE 2383 bus case, and a speedup of 9.4 for an artificial 4766 bus case made by a form of doubling the matrix sizes for the IEEE 2383 bus case.

In [76], the same authors as in [75] implement the same methods, only using the proprietary Jacket library of sparse matrix routines for GPGPUs. To generate large test cases, the IEEE 2383 bus case matrices were put through a form of doubling, tripling, and quadrupling to produce test cases of 4766, 7149, and 9532 buses. Results for the Newton-Raphson method were a speedup of 35 for the 2383, 4766, and 7149 bus cases. Results for the Newton-Raphson method were a speedup of 28 for the 7149 and 9532 bus cases.

6.3 Evaluating Power Flows in Parallel on the GPGPU

6.3.1 One Contingency Per Color Channel on a pre-CUDA GPU

In [71], the authors implemented (N-1) contingency analysis using a DC power flow on a pre-CUDA GPU, which oddly enough required all vectors to be re-mapped as matrices in order to be computed using graphics commands. The authors used a sparse matrix storage method that is not general-purpose, but specific to the properties of the DC power flow B matrix, exploiting the fact that in a typical power system the number of non-zero elements is not more than ten regardless of system size. Four contingencies at a time were computed in parallel, taking advantage of the GPU's ability to execute SIMD parallelism on the four color channel, red, green, blue, and transparency. Results were a speedup of 4 for both the IEEE 300 bus system and "a fictitious 1000 bus system based on assumptions of network sparsity, realistic serial and shunt admittance size and Y-bus symmetries".

6.3.2 One Power Flow Per GPGPU Thread Block

In [77] the authors applied existing GPGPU implementations of Jacobi's method to a power flow. The authors implemented a sparse storage scheme for the admittance matrix, made up of the diagonal elements stored in a vector Y^D, the remainder elements stored in a compact matrix Y^R, and the column indexes for the elements stored in another compact matrix I^R. One power flow is computed by each block of threads with the admittance matrix stored in shared memory. The test case was the IEEE 118 bus model, but different contingencies were not actually evaluated to produce the results – the same power flow calculation was simply repeated by each thread block. Results were that it took a CPU

implementation 25 minutes to solve the IEEE 118 bus base case power flow 65,000 times, but it took the GPGPU implementation 18.6 seconds, for a speedup of 81.

6.4 Summary

The results discussed above do not include a full (N-1) contingency analysis on a GPGPU, only on a pre-CUDA GPU using graphics commands to manipulate arithmetic. Excepting those who limited their studies to very small test systems, a fundamental problem the authors above had to grapple with was the handling of large sparse matrices. Those with access to recent, proprietary sparse matrix libraries claimed some of the most spectacular speedup results.

The method outlined in this thesis not only attacks the problem of running (N-1) contingency analysis in parallel on a GPGPU, it reduces the handling of sparse matrices to a bare minimum. This makes it much more accessible than methods that require expensive proprietary packages to run. However, the method proposed in this thesis does still have some sparse matrix operations, and some of the impressive results outlined above can be implemented for the sparse matrix operations in this method.

Chapter 7

Conclusions

A long-established principle in parallel programming, Amdahl's Law, predicts a maximum theoretical speedup of an application program that can be gained by executing it in parallel. It states that the theoretical maximum speedup is limited by the time it takes to perform the serial portion of the code that was not parallelized [78]. Gustafson's Law is based on the concept that Amdahl's Law doesn't take into account the way that most people behave. Only parallel programmers are fundamentally concerned with ever-finer gradations of speedup of code, whereas most users of application code simply want the best results that they can get within what they have determined to be a reasonable period of time [79]. As more processing power becomes available, they will want more and better results, but the reasonable period of time tends to remain fixed or based on dictates that have little to do with parallel programming.

Such is the case for both static and dynamic power systems security analysis. In systems planning, static security analysis must complete within the time allotted for the study, which may well be proceeding in parallel with acquiring rights-of-way and legislative permission to move forward. In operational planning, such as day-ahead security analysis, static security analysis must complete within the operational planning window, since the power system will operate whether the study has completed or not. In real-time operations

as discussed in Chapter 2, regulatory requirements may set the ”reasonable time frame” to be 30 minutes or less, but practical requirements for different control centers have determined the ”reasonable time frame” to be a few minutes for (N-1) contingency analysis, with more minutes added for selected (N-1-1) and other additional contingencies as time allows. The contingencies selected for study are limited by the time and computing power available. However, regulatory requirements are becoming more stringent and systems larger and more complex, creating an urgent need for more analysis of more contingencies to be completed in the minutes allotted.

Adding GPGPUs to the nodes in a control center's existing computational platform is a significantly lower expense than adding an equivalent number of new nodes and the infrastructure to support them, even if the nodes' power supplies must be upgraded to support the demands of the GPGPU. If accelerating computations with GPGPUs can halve the time needed for for a set of contingencies to run on a set of given computational nodes, freeing up crucial minutes for analysis of additional contingencies, the investment can be worth the costs. And yet this would be considered a quite modest speedup for GPGPU computing if the problem is conditioned in a way that maps well to the architecture and programming model of the GPGPU [57][10][60][61][62][63][64]. As we have seen in Chapter 6, halving the time needed for contingency analysis to complete can be accomplished simply by farming out only part of the power flow computation to the GPGPU in a reasonable manner and running the remaining parts of the computation on the CPU.

The novel method for GPGPU contingency analysis and its variants presented in this thesis allows that process of speedup to be taken substantially further, since it re-maps as

much of the computation as possible to be a series of dense vector operations based on simple arithmetic that is conservative with respect to data movement and flexible with respect to implementation details such as thread block size. Where sparse matrix operations cannot be avoided, this method, by slicing across contingencies, re-maps such operations to the much more efficient problem of a sparse matrix multiplied by a block of dense vectors larger than the matrix itself. The method applies to (N-1-1), (N-2), and (N-3) contingencies with little modification and little increase in computational burden or data movement per contingency.

Early implementations are promising. Some timing results for versions of two component routines are given in Appendix A. As discussed in Chapter 6, for small systems the true strengths of the GPGPU do not come into play. Small systems are insufficient to keep the GPGPGU occupied to any meaningful extent, however for realistically large systems the method developed in this research shows the highest benefits. This method is currently being implemented in continuing research by three members of the research team at the University of Minnesota led by Professors Amin and Wollenberg. Unprecedented in the body of work and applications that are currently available, this method is designed to accommodate systems of thousands to tens of thousands of buses, if need be, with the large power systems resulting from control area consolidation in mind.

Bibliography

[1] M. A. Alsaffar, "Voltage collapse and power flow algorithms," Ph.D. dissertation, Minneapolis, MN, USA, 2005.

[2] L. L. Grigsby, Ed., Power System Stability and Control, Third Edition (The Electric Power Engineering Handbook), 3rd ed. CRC Press, Apr. 2012.

[3] J. W. Pope, "Transmission reliability under restructuring," in Power Engineering Society Summer Meeting, 1999. IEEE, vol. 1. IEEE, Jul. 1999, pp. 162–166 vol.1.

[4] A. J. Wood and B. F. Wollenberg, Power Generation, Operation, and Control, 2nd ed. Wiley-Interscience, 1996.

[5] N. Balu, T. Bertram, A. Bose, V. Brandwajn, G. Cauley, D. Curtice, A. Fouad, L. Fink, M. G. Lauby, M. G. Lauby, B. F. Wollenberg, A10, J. N. Wrubel, and A11, "On-line power system security analysis," Proceedings of the IEEE, vol. 80, no. 2, pp. 262–282, 1992.

[6] G. C. Ejebe and B. F. Wollenberg, "Automatic contingency selection," IEEE Transactions on Power Apparatus and Systems, vol. PAS-98, no. 1, pp. 97–109, 1979.

[7] Q. Morante, N. Ranaldo, A. Vaccaro, and E. Zimeo, "Pervasive grid for large-scale power systems contingency analysis," Industrial Informatics, IEEE Transactions on, vol. 2, no. 3, pp. 165–175, Aug. 2006.

[8] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," IEEE Transactions on Computers, vol. C-21, no. 9, pp. 948–960, Sep. 1972.

[9] CUDA C Programming Guide, NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, USA., October 2012.

[10] NVIDIA CUDA Compute Unified Device Architecture Programming Guide 1.0, NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, USA., Jun. 2007.

[11] AMD Stream Computing User Guide, Advanced micro Devices, Inc., One AMD Place Sunnyvale, CA 94088-3453, Dec. 2008. [Online]. Available: http://ati.amd.com/technology/streamcomputing/Stream_Computing_User_Guide.pdf

[12] G. Constable and B. Somerville, Century of Innovation. National Academies Press, October 2003.

[13] K. Hamachi-LaCommare and J. Eto, "Understanding the cost of power interruptions to u.s. electricity consumers," Ernest Orlando Lawrence Berkeley National Laboratory, Tech. Rep., September 2004.

[14] D. Lineweber and S. McNulty, "The cost of power disturbances to industrial and digital economy companies," CEIDS, 1001 Fourier Drive, Suite 200, Madison, WI 53717, Tech. Rep., June 2001.

[15] S. Hoffman, "Power delivery system of the future: A preliminary estimate of costs and benefits," Electric Power Research Institute, Palo Alto, CA: 2004. 1011001., Tech. Rep., July 2004.

[16] "Estimating the costs and benefits of the smart grid," Electric Power Research Institute (EPRI), 3420 Hillview Avenue Palo Alto, CA 94304-1338, Tech. Rep., Mar. 2011.

[17] J. Thornton. (2003, May) The value of electricity when it's not available. NREL/BR-200-34231.

[18] R. Basu and J. M. Samet, "Relation between elevated ambient temperature and mortality: A review of the epidemiologic evidence," Epidemiologic Reviews, vol. 24, no. 2, pp. 190–202, December 2002.

[19] S. Whitman, G. Good, E. R. Donoghue, N. Benbow, W. Shou, and S. Mou, "Mortality in chicago attributed to the july 1995 heat wave." Am J Public Health, vol. 87, no. 9, pp. 1515–1518, September 1997.

[20] H. S. Ray, A. Bandopadhyay, and S. Chakrabarti, Energy Efficient and Environment Friendly Technologies for Rural Development (EETRD-2002) ; Proceedings of the Workshop Organized by the Millennium Institute of Energy and Environment Management, Kolk. Allied Publishers Pvt. Ltd., 2005.

[21] L. Wei and M. Zhang, "The multimodality of internet use: Demographic antecedents and political consequences," in Computer Engineering and Technology (ICCET), 2010 2nd International Conference on, vol. 4. IEEE, Apr. 2010, pp. V4–108–V4–112.

[22] D. Boisteanu, R. Vasiluta, A. Cernomaz, and C. Mucenica, "Home monitoring

of sleep apnea treatment: Benefits of intelligent CPAP devices," in Advanced Technologies for Enhanced Quality of Life, 2009. IEEE, Jul. 2009, pp. 77–80.

[23] Q. Huang, N. N. Schulz, A. K. Srivastava, and T. Haupt, "Distributed state estimation with PMU using grid computing," in Power & Energy Society General Meeting, 2009. PES & IEEE. IEEE, Jul. 2009, pp. 1–7.

[24] H. Wu and J. Giri, "PMU impact on state estimation reliability for improved grid security," in Transmission and Distribution Conference and Exhibition, 2005/2006 IEEE PES. IEEE, May 2006, pp. 1349–1351.

[25] I. Gorton, Z. Huang, Y. Chen, B. Kalahar, S. Jin, D. Chavarria-Miranda, D. Baxter, and J. Feo, "A High-Performance hybrid computing approach to massive contingency analysis in the power grid," in e-Science, 2009. Fifth IEEE International Conference on. IEEE, Dec. 2009, pp. 277–283.

[26] S. Jin, Z. H. Huang, Y. Chen, D. G. Chavarría-Miranda, J. Feo, and P. C. Wong, "A novel application of parallel betweenness centrality to power grid contingency analysis," in Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. IEEE, Apr. 2010, pp. 1–7.

[27] U. Wappler and C. Fetzer, "Software encoded processing: Building dependable systems with commodity hardware," in Computer Safety, Reliability, and Security, ser. Lecture Notes in Computer Science, F. Saglietti and N. Oster, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 4680, ch. 34, pp. 356–369.

[28] L. Lu, D. Hildebrand, and R. Tewari, “Zone-based data striping for cloud storage,” IBM Journal of Research and Development, vol. 55, no. 6, pp. 1:1–1:10, Nov. 2011.

[29] Z. H. Huang, Y. Chen, F. L. Greitzer, and R. Eubank, “Contingency visualization for real-time decision support in grid operation,” in Power and Energy Society General Meeting, 2011 IEEE. IEEE, Jul. 2011, pp. 1–7.

[30] “Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations,” U.S.-Canada Power System Outage Task Force, Tech. Rep., April 2004.

[31] Z. H. Huang and J. Nieplocha, “Transforming power grid operations via high performance computing,” in Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE. IEEE, Jul. 2008, pp. 1–8.

[32] “2012 state of reliability,” North American Electric Reliability Corporation (NERC), 3353 Peachtree Road NE Suite 600, North Tower Atlanta, GA 30326, Tech. Rep., May 2012.

[33] G. D. Irisarri and A. M. Sasson, “An automatic contingency selection method for On-Line security analysis,” Power Apparatus and Systems, IEEE Transactions on, vol. PAS-100, no. 4, pp. 1838–1844, Apr. 1981.

[34] G. C. Ejebe, H. P. Van Meeteren, and B. F. Wollenberg, “Fast contingency screening and evaluation for voltage security analysis,” Power Systems, IEEE Transactions on, vol. 3, no. 4, pp. 1582–1590, 1988.

[35] Y. Chen and A. Bose, "Security analysis for voltage problems using a reduced model," Power Systems, IEEE Transactions on, vol. 5, no. 3, pp. 933–940, 1990.

[36] A security analysis and optimal power flow package with real-time implementation in Northeast China Power System, 1993.

[37] A. O. Ekwue, A. M. Chebbo, M. E. Bradley, and H. B. Wan, "Experiences of automatic contingency selection algorithms on the NGC system," Power Engineering Review, IEEE, vol. 18, no. 3, pp. 53–54, 1998.

[38] O. Ceylan, H. Dağ, and A. Özdemir, "Parallel contingency analysis using differential evolution based solution for branch outage problem," in Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control, 2009. ICSCCW 2009. Fifth International Conference on. IEEE, 2009, pp. 1–4.

[39] C. Subramani, S. S. Dash, A. A. Bhaskar, M. Jagadeeshkumar, and K. Balaji, "Soft computing for voltage stability analysis and contingency ranking in power system," in Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT '09. International Conference on. IEEE, Dec. 2009, pp. 583–587.

[40] C.-L. Chang and Y.-Y. Hsu, "A new approach to dynamic contingency selection," Power Systems, IEEE Transactions on, vol. 5, no. 4, pp. 1524–1528, 1990.

[41] C. A. Castro and A. Bose, "Correctability of voltage violations in on-line contingency analysis," Power Systems, IEEE Transactions on, vol. 9, no. 3, pp. 1651–1657, 1994.

[42] Y. Chen and A. Bose, "An adaptive pre-filter for the voltage contingency selection function," Power Systems, IEEE Transactions on, vol. 5, no. 4, pp. 1478–1486, 1990.

[43] K. Nara, K. Tanaka, H. Kodama, R. R. Shoults, M. S. Chen, P. Van Olinda, and D. Bertagnolli, "On-Line contingency selection algorithm for voltage security analysis," IEEE Transactions on Power Apparatus and Systems, vol. PAS-104, no. 4, pp. 846–856, 1985.

[44] R. Fischl, T. Halpin, and A. Guvenis, "The application of decision theory to contingency selection," Circuits and Systems, IEEE Transactions on, vol. 29, no. 11, pp. 712–723, 1982.

[45] R. H. Chen, J. Gao, O. P. Malik, G. S. Hope, Wang, and Xiang, "Multi-contingency preprocessing for security assessment using physical concepts and CQR with classifications," Power Systems, IEEE Transactions on, vol. 8, no. 3, pp. 840–848, 1993.

[46] F. Albuyeh, A. Bose, and B. Heath, "Reactive power considerations in automatic contingency selection," IEEE Transactions on Power Apparatus and Systems, vol. PAS-101, no. 1, pp. 107–112, 1982.

[47] S. Vemuri and R. E. Usher, "On-Line automatic contingency selection algorithms," IEEE Transactions on Power Apparatus and Systems, vol. PAS-102, no. 2, pp. 346–354, 1983.

[48] G. Irisarri, D. Levner, and A. M. Sasson, "Automatic contingency selection for On-Line security analysis - Real-Time tests," IEEE Transactions on Power Apparatus and Systems, vol. PAS-98, no. 5, pp. 1552–1559, 1979.

[49] J. R. Dondeti, C. Yang, K. Trotter, A. Witmeier, and K. Sherd, "Experiences with

contingency analysis in reliability and market operations at MISO," in Power and Energy Society General Meeting, 2012 IEEE. IEEE, Jul. 2012, pp. 1–7.

[50] D. Chatterjee, J. Webb, Q. Gao, M. Y. Vaiman, M. M. Vaiman, and M. Povolotskiy, "N-1-1 AC contingency analysis as a part of NERC compliance studies at midwest ISO," in Transmission and Distribution Conference and Exposition, 2010 IEEE PES. IEEE, Apr. 2010, pp. 1–7.

[51] J. A. Huang, L. Loud, G. Vanier, B. Lambert, and S. Guillon, "Experiences and challenges in contingency analysis at Hydro-Quebec," in Power and Energy Society General Meeting, 2012 IEEE. IEEE, Jul. 2012, pp. 1–9.

[52] F. Garcia, N. D. R. Sarma, V. Kanduri, G. Nissankala, K. Gopinath, J. Polusani, T. Mortensen, and I. Flores, "ERCOT control center experience in using Real-Time contingency analysis in the new nodal market," in Power and Energy Society General Meeting, 2012 IEEE. IEEE, Jul. 2012, pp. 1–8.

[53] J. T. Ma, X. Liu, H. Sinha, J. Luciano, and V. Tsolias, "User experiences with contingency analysis at NSTAR," in Power and Energy Society General Meeting, 2012 IEEE. IEEE, Jul. 2012, pp. 1–8.

[54] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, Computer Graphics: Principles and Practice in C (2nd Edition), 2nd ed. Addison-Wesley Professional, Aug. 1995.

[55] D. Blythe, "Rise of the Graphics Processor," Proceedings of the IEEE, vol. 96, no. 5, pp. 761–778, May 2008.

[56] 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,” in Power and Energy Society General Meeting, 2011 IEEE. IEEE, Jul. 2011, pp. 1–8.

[57] D. B. Kirk and W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series), 1st ed. Morgan Kaufmann, Feb. 2010.

[58] M. D. Mccool, “Scalable Programming Models for Massively Multicore Processors,” Proceedings of the IEEE, vol. 96, no. 5, pp. 816–831, Apr. 2008.

[59] R. K. Sato and P. N. Swarztrauber, “Benchmarking the connection machine 2,” in Supercomputing ’88: Proceedings of the 1988 ACM/IEEE conference on Supercomputing. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 304–309.

[60] NVIDIA CUDA Best Practices Guide 2.3, NVIDIA, Jul. 2009.

[61] CUDA C Programming Guide PG-02829-001_v5.0, v5.0 ed., NVIDIA Corporation, Oct. 2012.

[62] J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, 1st ed. Addison-Wesley Professional, Jul. 2010.

[63] R. Farber, CUDA Application Design and Development, 1st ed. Morgan Kaufmann, Nov. 2011.

[64] NVIDIA CUDA Programming Guide PG-02829-001_v.21, NVIDIA, Dec. 2008.

[65] "IEEE Standard for Floating-Point Arithmetic," Microprocessor Standards Committee of the IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, Tech. Rep., Aug. 2008.

[66] O. Rosenberg, "OpenCL overview," Khronos Group, Tech. Rep., Nov. 2011.

[67] V. Volkov. (2010, Sep.) Better performance at lower occupancy. [Online]. Available: http://www.cs.berkeley.edu/\~{}volkov/volkov10-GTC.pdf

[68] D. M. Anderson and B. F. Wollenberg, "Power System Steady State Security Analysis Using Vector Processing Computers," Power Engineering Review, IEEE, vol. 12, no. 11, pp. 49+, 1992.

[69] A. Azzalini, "Matrix inversion lemma," August 2006.

[70] B. Stott and O. Alsac, "Fast decoupled load flow," Power Apparatus and Systems, IEEE Transactions on, vol. PAS-93, no. 3, pp. 859–869, May 1974.

[71] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units," in Power Tech, 2007 IEEE Lausanne. IEEE, Jul. 2007, pp. 731–736.

[72] N. Garcia, "Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach," in Power and Energy Society General Meeting, 2010 IEEE. IEEE, Jul. 2010, pp. 1–4.

[73] C. Guo, B. Jiang, H. Yuan, Z. Yang, L. Wang, and S. Ren, "Performance Comparisons of Parallel Power Flow Solvers on GPU System," in Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on. IEEE, Aug. 2012, pp. 232–239.

[74] A. Wigington, L. Min, C. Li, W. Murray, and A. Narayan, "Advancing the adoption of advanced computing methods and technologies for real-time control center operations," in Power and Energy Society General Meeting, 2012 IEEE. IEEE, Jul. 2012, pp. 1–6.

[75] J. Singh and I. Aruni, "Accelerating Power Flow studies on Graphics Processing Unit," in India Conference (INDICON), 2010 Annual IEEE. IEEE, Dec. 2010, pp. 1–5.

[76] ——, "Using jacket: Productivity platoform for gpu computing," AccelerEyes, 75 5th Street NW, Suite 204, Atlanta, GA 30308, Tech. Rep., 2010.

[77] C. Vilacha, J. C. Moreira, E. Miguez, and A. F. Otero, "Massive Jacobi power flow based on SIMD-processor," in Environment and Electrical Engineering (EEEIC), 2011 10th International Conference on. IEEE, May 2011, pp. 1–4.

[78] G. M. Amdahl, "Limits of expectation," International Journal of High Performance Computing Applications, vol. 2, no. 1, pp. 88–94, Mar. 1988.

[79] J. L. Gustafson, "Reevaluating amdahl's law," in Communications of the ACM, 1988, pp. 532–533.

Appendix A

Preliminary Results

The figures in this appendix show results for different component routines of this method for various system sizes including 32, 64, 128, 200, 256, 512, 1024, 2048, 4096, 8192, and 16384 buses.

As discussed in Chapter 6 and as noted in Chapter 7, for small systems the true strengths of the GPGPU do not come into play. This is also illustrated in Figure A.1 below. This figure shows timing results averaged over ten runs for vector itemwise multiplication (the vector inner product) for system sizes of 32, 200, 1024, 4096, 8192, and 16384 buses. The smallest size, 32, is insufficient to keep the GPGPGU occupied to any meaningful extent, and it is therefore slower than the next size of 200. For realistically large systems studied, GPGPGU methods show the highest benefits as illustrated in Figure A.1 – the computation time for the 8192 bus system was less than 1 percent higher than the time for the 4096 bus system. The time for the 16384 bus system goes up substantially since it runs into limits of the GPGPGU onboard memory, and data must be swapped out for the computation to complete. The onboard memory for the test system that produced these results was relatively low since this was run on the GeForce 8800 GTX, one of the first GPGPUs ever released, with hardware details given in Appendix B

Figure A.2 shows timing results averaged over ten runs for a sparse matrix multiplied

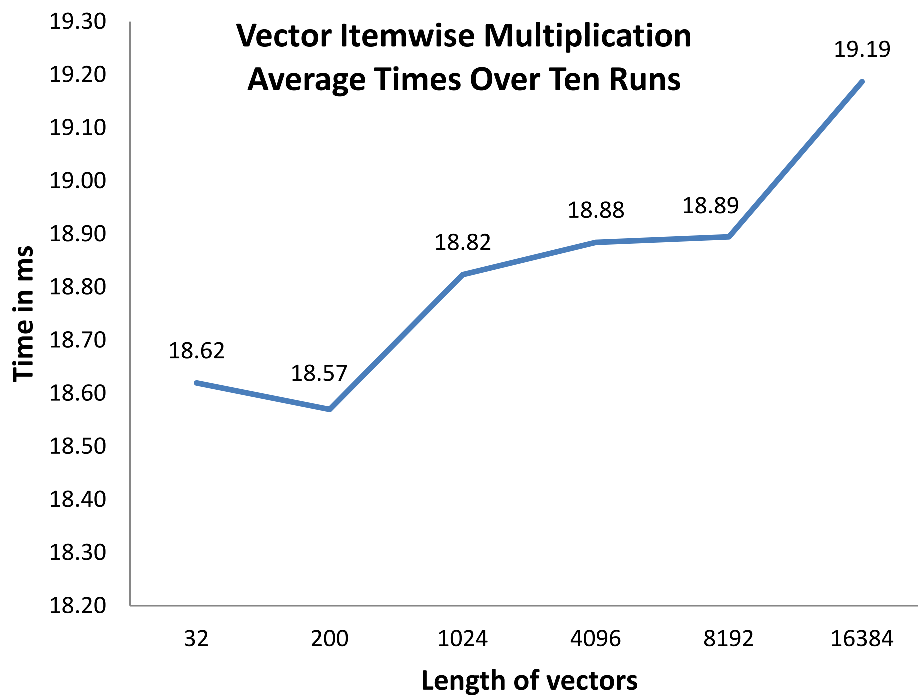

Figure A.1: Timing results averaged over ten runs for vector itemwise multiplication (the vector inner product) for system sizes of 32, 200, 1024, 4096, 8192, and 16384 buses.

by each of a set of dense vectors, where the system size N indicates the dimensions of the square sparse matrix, the length of the vectors, and the number of vectors. The systems sizes are 512, 1024, 2048, 4096, 8192. This sets of timing results was generated using the GeForce GTX 470 with hardware details given in Appendix B. Since this is a much more data-intensive routine, though the total amount of shared memory per block is higher for the GeForce GTX 470 than for the GeForce 8800 GTX, the limits of onboard memory are reached for a much smaller system size as can be seen in Figure A.2.

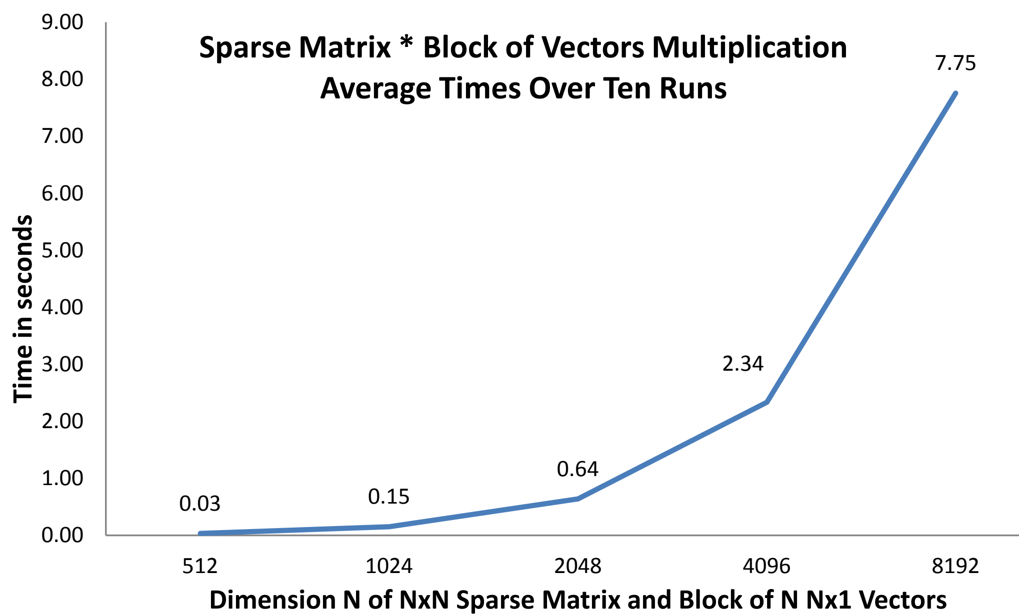

Figure A.2: Timing results averaged over ten runs for multiplication of a sparse matrix by a set of dense vectors for system sizes of 512, 1024, 2048, 4096, and 8192 buses

Appendix B

Compute Platform Details

Exact replication of the results given will depend on exact replication of the platform used. As much detail as possible is given here.

B.1 Software

All code for this project uses the CUDA Runtime API rather than the driver API.

- CUDA 3.1
- Ubuntu 9.04
- Kernel Linux 2.6.31-22-generic
- Compiler gcc 4.4

B.2 CPU Hardware

- Memory: 1.9GiB
- Processor 0: Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz
- Processor 1: Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz

B.2.1 Output of NVIDIA CUDA's deviceQuery

```
\$ ./deviceQuery

./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

There are 2 devices supporting CUDA

Device 0: "GeForce GTX 470"
CUDA Driver Version:                             3.10
CUDA Runtime Version:                            3.10
CUDA Capability Major revision number:           2
CUDA Capability Minor revision number:           0
Total amount of global memory:                   1341849600 bytes
Number of multiprocessors:                       14
Number of cores:                                 448
Total amount of constant memory:                 65536 bytes
Total amount of shared memory per block:         49152 bytes
Total number of registers available per block: 32768
Warp size:                                       32
Maximum number of threads per block:             1024
Maximum sizes of each dimension of a block:     1024 x 1024 x 64
```

```
Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
Maximum memory pitch:                          2147483647 bytes
Texture alignment:                             512 bytes
Clock rate:                                    1.22 GHz
Concurrent copy and execution:                 Yes
Run time limit on kernels:                     No
Integrated:                                    No
Support host page-locked memory mapping:       Yes
Compute mode:                                  Default (multiple
  host threads can use this device simultaneously)
Concurrent kernel execution:                   Yes
Device has ECC support enabled:                No


Device 1: "GeForce 8800 GTX"
CUDA Driver Version:                           3.10
CUDA Runtime Version:                          3.10
CUDA Capability Major revision number:         1
CUDA Capability Minor revision number:         0
Total amount of global memory:                 804585472 bytes
Number of multiprocessors:                     16
Number of cores:                               128
Total amount of constant memory:               65536 bytes
Total amount of shared memory per block:       16384 bytes
```

```
Total number of registers available per block: 8192
Warp size:                                      32
Maximum number of threads per block:            512
Maximum sizes of each dimension of a block:     512 x 512 x 64
Maximum sizes of each dimension of a grid:      65535 x 65535 x 1
Maximum memory pitch:                           2147483647 bytes
Texture alignment:                              256 bytes
Clock rate:                                     1.35 GHz
Concurrent copy and execution:                  No
Run time limit on kernels:                      Yes
Integrated:                                     No
Support host page-locked memory mapping:        No
Compute mode:                                   Default (multiple
  host threads can use this device simultaneously)
Concurrent kernel execution:                    No
Device has ECC support enabled:                 No

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 3.10,
CUDA Runtime Version = 3.10, NumDevs = 2,
Device = GeForce GTX 470, Device = GeForce 8800 GTX


PASSED
```

```
Press <Enter> to Quit...

-----------------------------------------------------------
```

B.2.2 Output of NVIDIA CUDA's bandwidthtest

```
\$ ./bandwidthTest

[bandwidthTest]

./bandwidthTest Starting...


Running on...


 Device 0: GeForce GTX 470

 Quick Mode


 Host to Device Bandwidth, 1 Device(s), Paged memory

   Transfer Size (Bytes) Bandwidth(MB/s)

   33554432 1536.1


 Device to Host Bandwidth, 1 Device(s), Paged memory

   Transfer Size (Bytes) Bandwidth(MB/s)

   33554432 1498.3


 Device to Device Bandwidth, 1 Device(s)
```

```
   Transfer Size (Bytes) Bandwidth(MB/s)

   33554432 94042.0



[bandwidthTest] - Test results:

PASSED



Press <Enter> to Quit...

-----------------------------------------------------------
```

B.2.3 Output of lspci

```
\$ lspci

00:00.0 Host bridge: nVidia Corporation C55 Host Bridge (rev a2)

00:00.1 RAM memory: nVidia Corporation C55 Memory Controller

   (rev a1)

00:00.2 RAM memory: nVidia Corporation C55 Memory Controller

   (rev a1)

00:00.3 RAM memory: nVidia Corporation C55 Memory Controller

  (rev a1)

00:00.4 RAM memory: nVidia Corporation C55 Memory Controller

   (rev a1)

00:00.5 RAM memory: nVidia Corporation C55 Memory Controller
```

```
   (rev a2)
00:00.6 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:00.7 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.0 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.1 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.2 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.3 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.4 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.5 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:01.6 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:02.0 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:02.1 RAM memory: nVidia Corporation C55 Memory Controller
   (rev a1)
00:02.2 RAM memory: nVidia Corporation C55 Memory Controller
```

```
   (rev a1)
00:03.0 PCI bridge: nVidia Corporation C55 PCI Express bridge
   (rev a1)
00:09.0 RAM memory: nVidia Corporation MCP55 Memory Controller
   (rev a1)
00:0a.0 ISA bridge: nVidia Corporation MCP55 LPC Bridge (rev a2)
00:0a.1 SMBus: nVidia Corporation MCP55 SMBus (rev a2)
00:0b.0 USB Controller: nVidia Corporation MCP55 USB Controller
   (rev a1)
00:0b.1 USB Controller: nVidia Corporation MCP55 USB Controller
   (rev a2)
00:0d.0 IDE interface: nVidia Corporation MCP55 IDE (rev a1)
00:0e.0 IDE interface: nVidia Corporation MCP55 SATA Controller
   (rev a2)
00:0e.1 IDE interface: nVidia Corporation MCP55 SATA Controller
   (rev a2)
00:0e.2 IDE interface: nVidia Corporation MCP55 SATA Controller
   (rev a2)
00:0f.0 PCI bridge: nVidia Corporation MCP55 PCI bridge (rev a2)
00:0f.1 Audio device: nVidia Corporation MCP55 High Definition
   Audio (rev a2)
00:11.0 Bridge: nVidia Corporation MCP55 Ethernet (rev a2)
00:12.0 Bridge: nVidia Corporation MCP55 Ethernet (rev a2)
00:13.0 PCI bridge: nVidia Corporation MCP55 PCI Express bridge
```

```
   (rev a2)
01:00.0 VGA compatible controller: nVidia Corporation G80
   [GeForce 8800 GTX] (rev a2)
02:07.0 FireWire (IEEE 1394): Texas Instruments TSB43AB22/A
   IEEE-1394a-2000 Controller (PHY/Link)
03:00.0 VGA compatible controller: nVidia Corporation Device
   06cd (rev a3)
03:00.1 Audio device: nVidia Corporation Device 0be5 (rev a1)
```