Fast Security Constraint Optimal Power Flow using Parallel and Heterogenous Computing

Diego Fernando Rodríguez Medina

Universidad Nacional de Colombia
Department of Electrical Engineering
Bogotá, Colombia
2021

Fast Security Constraint Optimal Power Flow using Parallel and Heterogenous Computing

Diego Fernando Rodríguez Medina

A thesis submitted in partial fulfillment of the requirements for the degree of:
Doctor of Engineering

Thesis Director:
Sergio Rivera, PhD

Thesis Committee:
Di Wu, PhD
Jaime Pinzón, PhD
Marcelo Elizondo, PhD

Research line: Power System Optimization

Universidad Nacional de Colombia
Department of Electrical Engineering
Bogotá, Colombia
2021

Agradecimientos

Hay pocos momentos en los que no es fácil comunicar las muchas cosas que pasan por la cabeza. Sin embargo, trataré de hacerlo en unas cortas líneas. Primero desearía agradecer al profesor Sergio Rivera que más que un director ha sido un amigo durante más de seis años de relación. De esta misma forma, agradecerle al comité de jurados (Dr. Di Wu, Dr. Jaime Pinzón, Dr. Marcelo Elizondo), que de manera desinteresada y amable han optado por leer las páginas que se registran en este documento, dando las mejores recomendaciones y sugerencias. Además de esto un gran agradecimiento a mi amiga durante estos últimos años, Gabriela Andrade "Pequeño Iceberg" que me ha ayudado enormemente con la edición de la tesis y acompañado en más de un momento difícil de estos últimos años.

Es imposible evitar nombrar a las personas especiales que han estado acompañándome durante toda mi vida, como son mis padres Jorge y Flor que han formado lo que hoy soy. A mi hermano que, a pesar de ser tan diferentes, amo de forma inimaginable. A mi segunda mamá que es mi Tía Naty, a quién siempre llevaré en mi corazón y a quién sencillamente debo mi personalidad. De igual forma unas gracias especiales a mi amigo desde el colegio William Mejía, con el que hemos sorteado un gran número de retos, los cuales, afortunadamente, no han logrado vencernos. A mis amigos de la Universidad, en especial Andrés Angulo que ha estado día y noche apoyándome en las labores que debo desarrollar, a Carlos Ávila y a todos los que no hemos perdido contacto a pesar de la distancia (Camilo, David, Fernando, Sebastián, Ruben, Carlos M, Rafa). También a mis amigos de colegio (David, Alejandro, Peter, Néstor, Diego, Ángela, Laura) con los que siempre hemos sido una familia. A Ruth que siempre ha estado orgullosa de cada uno de mis logros y ha seguido cada uno de ellos. A Diego Gómez y Wilmer Garzón que más que un apoyo en mi equipo de consultoría, han sido unos grandes compañeros y co-equiperos en todo el desarrollo de esta tesis. Podría decir que gracias a ellos no perdí la esperanza de finalizar este documento. Es innegable agradecer a mi equipo de GERS que ha sido una pieza fundamental en este proceso (Juan G., Victoria G., Carolina R., Jose H., Cristhian G., Daniela Z., Daniel S., Andrés Z., Daniel G., Manuel T., Manuel F., Ricardo A., David A. y todos los que pasaron por la escuela de Estudios Internacionales).

Por último, pero no menos importante, un agradecimiento gigantezco a mi especial "Maracaibo" quién sabe que llevo en mi corazón y que alegró más de un momento en este proceso. Ella sabe que puede alegrarme muchos más, sólo que no todo es tan fácil en este mundo desordenado. Ella sabe cuánto sueño con tenerla a mi lado, la adoro.

A todos un especial y caluroso agradecimiento. Soy consciente que mi vida no sería lo mismo sin ustedes.

Acknowledge

There are few moments when it is not easy to communicate the many things that go through your head. However, I will try to do it in a few short lines. First, I would like to thank Professor Sergio Rivera, who more than a director has been a friend for more than six years of relationship. In the same way, thank the committee members (Dr. Di Wu, Dr. Jaime Pinzón, Dr. Marcelo Elizondo), whose in a disinterested and friendly way have chosen to read the pages that are recorded in this document, giving the best recommendations and suggestions. In addition to this, a big thank you to my friend in recent years, Gabriela Andrade "Little Iceberg" who has helped me enormously with the thesis edition and accompanied me in more than one difficult moment of this period.

It is impossible to avoid naming the special people who have been accompanying me throughout my life, such as my parents Jorge and Flor who have formed what I am today. To my brother who, despite being so different, I love in an unimaginable way. To my second mother who is my Aunt Naty, whom I will always carry in my heart and to whom I simply owe my personality. In the same way, a special thanks to my friend from school William Mejía, with whom we have overcome a large number of challenges. To my friends from the University, especially Andrés Angulo who has been supporting my days and nights at GERS, Carlos Ávila and all who keep in touch despite the distance (Camilo, David, Fernando, Sebastián, Ruben, Carlos M, Rafa). Also to my friends from school (David, Alejandro, Peter, Néstor, Diego, Ángela and Laura) with whom we have always been a family. To Ruth who has always been proud of each of my accomplishments and followed all of them. To Diego Gómez and Wilmer Garzón who, more than a support in my consulting team, have been great colleagues throughout the development of this thesis. I could say that thanks to them I did not lose hope of finalizing this document. It is undeniable to thank my GERS team that has been a fundamental piece in this process (Juan G., Victoria G, Carolina R., Jose H., Cristhian G., Daniela Z., Daniel S., Andrés Z., Daniel G., Manuel T., Manuel F., Ricardo A., David A. and all those who went through the school of International Studies)

Last but not least, a special thanks to my special "Maracaibo" who knows what is in my heart. She made me happy in more than one moment during all the years of this work. She can keep me happy even in this not well organized world. "La adoro" and dream it.

To all a special and warm thanks. I am aware that my life would not be the same without you.

Abstract

Optimal and secure grid operation is paramount for modern power systems. However, the ever increasing system size, number of conventional and renewable sources, not to mention system loads and power system controllers, make the satisfaction of those requirements in on-line applications not an easy task. Different approaches have been applied to meet power system security criteria and reach optimal cost during real-time operation. Nevertheless, the strategies are mostly employed in small power systems, using strong assumptions or lack of advanced and efficient software-hardware interaction. That makes some of the applications infeasible in real operation or very costly in terms of hardware implementation. As a solution for those limitations, this research will address the problem of Security Constrained Optimal Power Flow (SCOPF) using the potential of Parallel and Heterogeneous Computing (PHC). By this approach, this research is looking to expand the application of advanced computing techniques for the solution of real-time power system problems that simultaneously involves security and optimal cost. The intention is to understand the strategies and principles for computer memory management, data structures and SCOPF re-formulation to optimally satisfy security and time response for proper power system operation.

Keywords: Security Constrained Optimal Power Flow (SCOPF), Optimal Power Flow (OPF), Parallel Computing (PC), Complex Power Networks, Real-Time SCOPF, Graphical Processing Unit (GPU).

Resumen

Cálculo Rápido de Flujo de Potencia Óptimo con Restricciones de Seguridad utilizando Computación Paralela y Heterogénea

El funcionamiento óptimo y seguro de la red eléctrica es primordial para los sistemas de energía modernos. Sin embargo, el tamaño cada vez mayor de dichos sistemas, así como la cantidad de fuentes convencionales y renovables, sin mencionar las cargas del sistema y los controladores del sistema de energía, hacen que la satisfacción de esos requisitos en las aplicaciones en tiempo real no sean una tarea fácil. Se han aplicado diferentes enfoques para cumplir con los criterios de seguridad del sistema de energía y alcanzar un costo óptimo durante la operación en tiempo real. Sin embargo, las estrategias se emplean principalmente en sistemas académicos de pequeñas dimensiones, utilizando fuertes suposiciones o falta de software-hardware avanzado y eficiente interacción. Eso hace que algunas de las aplicaciones sean inviables en operación real o muy costosas en términos de implementación de hardware. Como una solución para esas limitaciones, esta investigación abordará el problema del flujo de energía óptimo con restricciones de seguridad (SCOPF) utilizando el potencial de la computación paralela y heterogénea (PHC). Mediante este enfoque, esta investigación busca expandir la aplicación de técnicas informáticas avanzadas para la solución de problemas de sistemas de potencia en tiempo real que involucran simultáneamente seguridad y costo óptimo. La intención es comprender las estrategias y los principios para la gestión de la memoria de la computadora, las estructuras de datos y la reformulación de SCOPF para satisfacer de manera óptima la seguridad y el tiempo de respuesta para correcto funcionamiento del sistema de potencia.

Palabras Clave: Flujo de Potencia Optimo, Seguridad en Sistemas de Potencia, Computación Paralela, Redes de Potencia Complejas, Operación en Tiempo Real, Unidad de Procesamiento Gráfico .

Contents

List of Figures

List of Tables

Acronyms

ADMM Alternating Direction Multipliers Method. 48, 51, 52

AGC Automatic Generation Control. 76, 77

ALM Augmented Lagragian Method. 48, 51, 52

ALU Arithmetic Logic Unit. 16

BD Benders Decomposition. 48, 51, 52, 74

BESS Battery Energy Storage Systems. 27

CISC Complex Instruction Set Computing. 16

COO Sparse Coordinate Format. 81

CPU Central Processing Unit. ix, 8, 9, 15, 16, 18, 19, 22, 24, 25, 27–31, 34, 39, 44, 75, 76, 78, 80, 81, 83, 85–89, 91, 94, 96–98

CS Contingency Screening. ix, 68, 71, 72, 74

CSCOPF Corrective Security Constrained Optimal Power Flow. 14, 50, 51

CSR Compressed Sparse Row Format. 40, 81

CUDA Parallel Computing Platform and Programming model developed by NVIDIA. 81, 83, 85, 94

DAEs Differential-Algebraic Equations. 22, 25

DC Direct Current. 22, 50–52

DE Differential Evolution. 31

DEC Decoupled Power Flow. 24

EA Evolutionary Algorithm. 52

EC Embedded Computer. 8, 33, 34, 36, 39, 42, 44, 47

EMS Energy Managment System. 11

EMT Electromagnetic Transient. 22, 27, 28

FERC Federal Energy Regulatory Commission. 49

FPGA Field Programmable Gate Array. 15, 21, 22, 25, 27, 28, 31

FSM Finite State Machine. 64, 65, 71

GA Genetic Algorithm. 52

GPGPU General Purpose Graphics Processing Unit. 18, 19

GPU Graphical Processing Unit. v, ix, 8, 9, 15, 18, 19, 21–25, 28–31, 34, 39, 40, 42, 44, 47, 75, 76, 78, 80, 81, 83, 85–89, 91, 94, 96–98

HC Heterogeneous Computing. 15, 19, 96

HPC High Performance Computing. 9, 15, 31, 74, 96

HVDC High Voltage Direct Current. 21

IAI Interlaced Alternating Implicit. 25

IED Intelligent Electronic Devices. 34

ILP Instruction Level Parallelism. 18

IPOPT Interior Point Optimization. 28, 31, 60, 61, 68

ISA Instruction Set Architecture. 16

LD/ST Load/Store units. 19

LDAG Layered Directed Acyclic Graph. 22

LEM Local Energy Markets. 98

MA Metaheuristic Algorithm. 52

MILP Mixed Integer Linear Programming. 48

MIMD Multiple Instruction, Multiple Data. 21

ML Machine Learning. 52

MM Metaheuristic Method. 28, 30

MPI Message Passing Interface. 21, 25, 27, 31

NERC North American Electric Reliability Corporation. 49

NNZ Non Zeros. 28

NP-Hard Nonlinear Programming Problems-Hard. 50

OPF Optimal Power Flow. v, vii, ix, 8, 13, 24, 27–32, 34, 49, 51, 52, 60, 65, 74, 76, 78, 88, 89, 97, 98

PC Parallel Computing. v, 9, 15, 21, 22, 25, 27, 31, 76, 87, 96, 98

PCI Peripheral Component Interconnect. 83

PDC Parallel and Distributed Computing. 15

PEC Personal Computer. 33, 34, 44, 47

PF Power Flow. 49

PHC Parallel and Heterogeneous Computing. v, vii, ix, 6, 8, 9, 13, 18, 21, 23–29, 31, 75, 76, 80, 87–89, 91, 94, 97

POW Point of Wave. 31

PSCOPF Preventive Security Constrained Optimal Power Flow. 14, 49, 50

PSO Particle Swarm Optimization. 24, 27, 28, 31

QC Quatum Computing. 15

RAM Random Access Memory. 16

RISC Reduce Instruction Set Computer. 15, 16

SCOPF Security Constrained Optimal Power Flow. v, vii, 8, 9, 12, 13, 25, 27–29, 31, 48–53, 58, 60, 68, 71, 74–78, 80, 81, 87–89, 91, 94, 96–98

SFU Special Function Units. 19

SIMD Single Instruction, Multiple Data. 21, 22, 24, 25

SIMT Single Instruction Multiple Threads. 6

SLP Successive Linear Programming. 51

SM Streaming Multiprocessors. 19, 83

SP Streaming Processor. 83

SP Scalar Processor. 19

TC Tensor Cores. 19

TEF Transient Energy Function. 31

TS Transient Stability. 22, 25, 28, 31

TSCOPF Transient Stability-Constrained Optimal Power Flow. 25, 27, 28, 31

ULM Universal Line Models. 22

UMM Unified Machine Models. 22

1 Introduction

1.1 Motivation

Since the start of electric power systems, electrical engineers and system operators have managed the grid in real-time, balancing the energy produced by generation plants and the demand consumed by different customers. Nowadays, the increase of energy sources and loads have expanded the complexity of this labor, so numerous intelligent systems and sophisticated solutions such as Energy Managment Systems (EMS), Dynamic Stability Assessment (DSA) tools, Market Management Systems (MMS), Advanced Distribution Management Systems (ADMS), among others are required to efficiently and securely operate the vast amount of system elements.

Previous solutions not only monitored and operated electrical elements but also ran algorithms and routines using Optimal Power Flow (OPF) models to satisfy optimal system operation and control in a normal state. The objective of the OPF includes finding optimal generation dispatch and control settings in the machines, without exceeding limits in the grid elements to maximize one or more objectives [5]. Nevertheless, optimal operation during normal state is not enough in our society with a high dependence on electricity, making the desire of maintaining system security an overriding factor. System security follows a planning and operation criteria that keep the system operating within secure limits after the failure of one or more elements [6, 7]. In response to those requirements, Security-Constrained Optimal Power Flow (SCOPF) formulation allows for an optimal solution satisfying security constraints. These constraints involve the loss of a line, a generation unit, or an element that may cause power interruption or abnormal operation [8].

Several general purpose optimization solvers and solution methods have been implemented to find SCOPF problem solutions in the last era, however limitations in computer capabilities and technology cost resulted in limited optimal solutions. The main reason is that the current power system has expanded considerably in the last century [9], increasing the number of lines, nodes, loads, and generation units in the grid thus making the system more complex. Trying to solve these ever increasing problems, grid operators and the research public were required to make model simplifications to satisfy effective time responses in the control center. The assumptions included linearizing AC power flow

formulation (using Decouple Optimal Power Flow (DCOPF) to ignore voltage and reactive power in the system, avoiding discrete system changes, removing systems controls, etc., to identify feasible solutions with an efficient cost and time [10].

There are several technology and OPF formulation advances that hypothesize that the entire SCOPF problem can be solved in a useful time for grid operators with an improved system cost. Computer power boost [11], new SCOPF approaches, and grid operation techniques as well as computer accessibility, exalt the previous premise.

As mentioned above, different approaches have been chosen to use developments in PHC in order to efficiently use the resources available in a computer or in large-scale clusters with hundreds of nodes. Most of these problems have identified great potential in solving problems focused on changes in power flow and transient stability [12–16]. A vast majority of approaches have used parallel computing, mostly using computer CPU cores, however, the high cost, and high temperatures generated during software processing, low latency of memory access, as well as barriers to the simple execution of instructions in parallel has limited the massive use of this alternative. On the other hand, other approaches have opted for cloud processing or the use of GPUs, the latter being one of the most used in handling high volumes of information with regular processing patterns due to the benefits shown at low cost.

The use of a GPU allows for the processing of a large number of homogeneous operations, with a reduced time due to the thousands of cores and the Single Instruction Multiple Threads (SIMT) structure it handles. On the other hand, the CUDA language, developed in previous years [17, 18], makes interaction with this type of hardware more flexible. These and other benefits make using GPUs for highly computationally and complex problems a feasible solution in terms of cost and response times. However, its benefits are limited by the requirements of regular computing patterns [18].

In order to use the PHC potential and increase the computational capacity applied to power systems, ensuring optimal, secure operation, as well as fast processing, this work proposes the SCOPF implementation using the integration of the applied CPU and GPU potential to SCOPF solutions.

1.2 Research Statement

The increase of electrical energy consumption and dependence has motivated power system operation to function optimally and securely. Nowadays, operators must guarantee constant and efficient energy flows not only in normal states but also in emergency

conditions, with one or more elements out of service. However, this requirement is not easily satisfied as a result of the number of variables and system sizes involved in the power systems. Controllable and non-controllable variables include loads, switchable capacitors and reactors, conventional and renewable sources, and breakers. System sizes depend on the number of substations, transmission lines, and transformers. A greater number of variables and system sizes results in extreme memory requirements and unacceptable computation times for SCOPF, as a result of high dimensionality, non-linearity, and non-convexity of the problem.

Multiple strategies have been implemented to solve SCOPF problems satisfying accuracy and calculation times required by grid codes. These strategies rarely are applied to high size systems (more than 1,000 nodes) and use complete problem definition. In contrast, most have been applied to small and medium size systems based on simplifications of the problem definition [19]. The simplifications include contingency filtering [20, 21], network compression [19] and linearization using DC flow [22], among others. Although these simplifications allow us to satisfy computation times and optimal solutions demanded by the operators, they still require robust and high processing platforms with an excessive number of cores or extreme approximations that affect the performance of the solution algorithms and the cost of the solution. Some technology advances have shown outstanding results with the use of small size PHC architectures with large data volumes and homogeneous processing; however, the use of these strategies have not been fully exploited in the solution of SCOPF problems.

In order to reduce the current limitations associated with the number of problem simplifications and excessive calculation times, advances in the area of PHC are proposed as a possible solution. However, new problem formulations will be required to reduce the gap between SCOPF and the use of CPU and GPU hardware. Through the research we will focus on the solution of SCOPF problems for medium and large power system using the PHC architectures. The research plans to answer the following questions:

- What specific SCOPF features and formulations are suitable for being parallelized using PHC architectures?
- How much faster can the evaluation be completed using PHC integration to solve SCOPF problems in comparison with a traditional solution using only CPU cores?
- Does the strategy satisfy the on-line security times and optimum solutions required by system operators?

1.3 Objectives

1.3.1 General Objective

Design a SCOPF algorithm using PHC in order to improve computation time and near optimal solution for on-line applications.

1.3.2 Specific Objectives

1. Identify main SCOPF features and formulations suitable for being parallelized in PHC architectures satisfying on-line security time limits.

2. Propose a SCOPF formulation using a PHC architecture to speed up near optimal solutions in Medium and Large Scale Power Systems.

3. Assess the performance of the algorithm with Medium and Large Scale Power Systems in on-line time frames.

1.4 Thesis Outline

The dissertation is divided in the set of chapters below. Chapter 3 supports **Objective 1**. Chapter 3 and 4 support **Objective 2**. Chapter 5 supports **Objective 3**.

1.4.1 Chapter 2: Background and Literature Review

This chapter presents a literature review covering the topics to be covered in the dissertation. First, the formulation of the Security Constrained Optimal Power Flow (SCOPF) problem is presented and the possible strategies that have been developed to solve this problem are shown. Subsequently, a review of the Parallel and Heterogeneous Computing (PHC) architectures currently offered by the market is made. A review of parallel structures applied to power systems and particularly to the Optimal Power Flow (OPF) and SCOPF solution is included. Finally, an in-depth review is done on the PHC application using Central Processing Unit (CPU) and Graphical Processing Unit (GPU).

1.4.2 Chapter 3: Parallel Power Flow Algorithm in Low Cost Embedded Computer Architectures Empowered by GPU

In this chapter, an algorithm based on the Newton-Raphson is vectorized and executed in a low-cost EC. The algorithm optimizes the execution time vectorizing Y_{bus} matrix, Jacobian matrix, and power injected flows using a GPU architecture. The algorithm is evaluated in different test power systems using Python and a computing board NVIDIA Jetson Nano.

The results show a feasible implementation for near real time power system planning and operation.

1.4.3 Chapter 4: Security Constraint Optimal Power Flow Formulation and Solutions with Constraint Handling

This chapter presents an implementation of SCOPF using a constraint handling strategy together with PC architecture to solve medium, large and complex power system in near real time. The strategy identifies the activities that consume higher times and uses special PC architectures to solve the problem in 5 and 45 minutes. The strategy was tested in power system with sizes from 500 to 11,615 buses in local computer and higher system sizes in ARPA computer cluster.

1.4.4 Chapter 5: SCOPF for Medium and Large Power Systems

This chapter shows the result of using a CPU and GPU based PHC architecture using the previously developed algorithm to solve SCOPF problems in medium, long and complex networks. The structure optimizes the tasks that can be parallelized in order to reduce the algorithm's execution times using CPU cores and GPU threads. The results were successful for large networks.

1.4.5 Chapter 6: Conclusions and Contributions

This chapters wrap up the concluding remarks of this research and proposes some research questions as future work. The key research questions presented in Section 1.2 are also addressed in this chapter.

1.5 Publications

The list of documents below were published during the development of this thesis:

- A Fast Decomposition Method to Solve a Security Constrained Optimal Power Flow (SCOPF) Empowered by High Performance Computing (HPC) *(Under Review)* [23].
- Low-Cost Analysis of Load Flow Computing Using Embedded Computer Empowered by GPU [24].
- A Review of Parallel Heterogeneous Computing Algorithms in Power Systems [25].
- Smart Microgrids Operation Considering Expert Knowledge and Ensembled Based Metaheuristic Optimization Algorithms *(Under Publication)* [26].

- A Fast Decomposition Method to Solve a Security-Constrained Optimal Power Flow (SCOPF) Problem Through Constraint Handling [27]
- Algorithms for Bidding Strategies in Local Energy Markets: Exhaustive Search through Parallel Computing and Metaheuristic Optimization [28].
- Teaching using a synchronous machine virtual laboratory [29].
- Smart Microgrids Operation Considering a Variable Neighborhood Search: The Differential Evolutionary Particle Swarm Optimization Algorithm [30].
- Mathematical Formulation and Numerical Validation of Uncertainty Costs for Controllable Loads [31].
- 2018 Grid Optimization Competition Evaluating the Performance of Modern Heuristic Optimizers on Stochastic Optimization Problems applied to Smart Grids Test bed A : Stochastic OPF in Presence of Renewable Energy and Controllable Loads [32].
- 2018 Competition on Operational Planning of Sustainable Power Systems: Testsbeds and Results [33].

1.6 Awards

- A Fast Decomposition Method to Solve Security Cosntrained Optimal Power Flow Empowered by Parallel and Heterogeneous Computing. Premio ÁMBAR a la Investigación 20/21 [34].
- Preventive Security Constrained Optimal Power Flow Using An Ensembled Method: Constraints Relaxation With Analitycal Optimization Combined With A Heuristic Method. ARPA Award. [35].
- Posicionamiento Óptimo de cuadrillas basado en estadísticas de Tránsito de Google Maps e Indicadores de Confiabilidad. Finalista Premio ÁMBAR a la Investigación 2018 [36].
- Herramienta para la Programación de Redes Inteligentes con Recursos Energéticos de Alta Incertidumbre. Finalista Premio ÁMBAR a la Innovación 2019 [37].

2 Background and Literature Review

Power systems require decisions that allow for their secure and optimal operation. Optimal operation is defined under lower cost criteria. On the other hand, security is defined as the ability of a power system to withstand sudden disturbances such as short circuits and the anticipated loss of system components [38]. These two characteristics mentioned above result from the requirements of operating manuals and network codes [39], in addition to the regularization of [40] markets.

Traditionally, power systems have been designed and operated to withstand contingency conditions of existing elements. According to this, network codes require that power systems be designed and operated in compliance with *disturbance-performance criteria*, supporting abnormal operating situations such as contingencies. The latter can include the output of one or more elements. Those outages are known as N-1 and N-K criteria [8] and are required to ensure that the system continues to operate normally in the face of unexpected variations.

Furthermore, due to the existence of economic transactions during the normal operation of the power system, it is necessary to optimize the operation costs, which is mainly achieved through the reduction of generation costs [5]. The optimization of operation costs has increased in complexity due to the wide variety of types of generations in the system, including hydraulic plants, fossil fuel dependent plants, and others that are booming. This is the case for both solar and wind renewable energies [41]. Figure **2-1** shows the different states that involve the operation of a power system [1]. In *Normal State*, the system will satisfy the normal operating ranges and ensure that it operates optimally. In this state, the connected load (equality constraint) must be met in the system without having violations of the thermal limits of lines and stresses (inequality constraint) throughout the system. A more extensive review of these constraints is observed in 2.1. Since the monitoring systems of the following system EMS can normally execute multiple assessments of possible contingency states, scenarios can be detected where a possible output of the system element can cause load loss, overload on lines, or cascading effects that could lead to blackouts. Such a state of operation is called the *Alert State*. In this state, the operator receives alarms of possible conditions that may generate some risk. In order to reduce the risk, maneuvers to avoid unsafe conditions are executed (In this condition, equality and

inequality constraint are maintained[1]). The system can reach *Emergency State* or *In Extremis* when a disturbance severe enough to leave an area without power or violate the safe voltage operating limits and line loadability (In this state the constraint of equality and inequality are not satisfied). Once this condition has been identified, the system operator must reduce violations that occurred and start the recovery process. This process is called *Restorative State*.

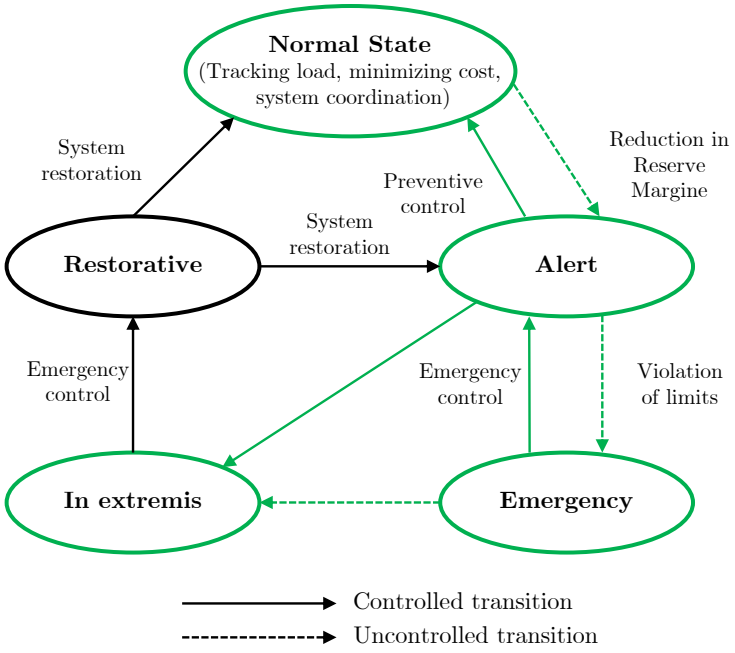

Figure 2-1: Power system operating states [1]

In order to increase system security, reduce the existing risks of load loss, line overhead, and cascading events, some approaches [42–46] have suggested strategies to cover the states highlighted in green in Figure **2-1**. In this way, not only is the optimal operation of the system guaranteed in the normal state of operation, but also in the emergency and extremis states. This includes the loss of loads and violations in the operating variables of various elements of the system, which are avoided. Studies focused on these solutions are framed in the area of SCOPF. These are based on optimal system operation in compliance with existing constraints during the normal operating and contingency [45] states.

Although these strategies [45] consider the green states of Figure **2-1**, they have been applied mostly to operation planning. This has culminated from the long processing times resulting from the large number of decision variables and constraints in the system. The

[1]In *Alert State* the occurrence of contingencies are not considered. This state appears when the system goes out of a certain security level or adverse weather conditions appear

SCOPF problems require the intensive use of computational resources and considerable extensions of evaluation time. This limits your application to tools that require short times, such as on-line security applications, such as the on-line security[2].

As a result of the above, evaluations in small-scale power systems or with simplifications in formulation have been proposed [42, 47, 48] in order to meet the limitations of computation and time. This suggests that a review and modification to the existing formulation of the existing formulation SCOPF and the computational architecture of PHC applied to online security could allow the desired time and optimal requirements to be met. The sections below describe the formulation of the problem SCOPF problema, a brief description of the computing architectures, and shows the advances of those applications to power systems.

2.1 Security Constraint OPF

The SCOPF problems are an extension of the de OPF[3], SCOPF problems can generally be formulated as shown below [45]:

$$\min_{x_0,...,x_c,u_0,...,u_c} \quad f_0(x_0,u_0) \tag{2-1a}$$

$$\text{s.t.} \quad g_0(x_0,u_0)=0, \tag{2-1b}$$

$$h_0(x_0,u_0)\leq L_l, \tag{2-1c}$$

$$g_k^s(x_k^s,u_0)=0 \qquad k=1,...,c, \tag{2-1d}$$

$$h_k^s(x_k^s,u_0)\leq L_s \qquad k=1,...,c, \tag{2-1e}$$

$$g_k(x_k,u_k)=0 \qquad k=1,...,c, \tag{2-1f}$$

$$h_k(x_k,u_k)\leq L_m \qquad k=1,...,c, \tag{2-1g}$$

$$|u_k-u_0|\leq \overline{\Delta u_k} \qquad k=1,...,c \tag{2-1h}$$

Where f_0 is the targeted function, k corresponds to the pre-contingency and post-contingency scenarios ($k = 0$ pre-contingency and c corresponds to the post-contingency scenario number c), x_k is the vector of state variables (magnitudes and voltage angles), x_k^s is the vector in the short-time frame (before the technical maneuvers made by the operator), u_k is the vector of control variables (generator power, voltages in terminals, shunt reactive compensators, transformer taps and pass shifter, breakers status), $\overline{\Delta u_k}$ is the vector of maximum allowed adjustments. L_l, L_m and L_s are the long (normal), medium, and short (emergency) term limits. Limits are different depending on the

[2]On-line and Real-time terms will be used indistinctly for time frames of less than an hour. This time frame includes generator droop responses and does not intend to cover in-deep analysis of Transient States

[3]Some authors consider that given the extension OPF is a section of SCOPF

maximum limits and constraint time.

Constraints 5-1b and 5-1c denote contingencies of pre-contingency states, 5-1d - 2-1h post-contingency states. Equality constraints 5-1b, 5-1d and 5-1f include nodal balances to meet the load. 2-1h allows the operator to make adjustments to the system after a contingency has happened. Due to the definition of the problem, two possible SCOPF formulations are considered:

- **Preventive Security Constrained Optimal Power Flow (PSCOPF)**: is a particular formulation of the SCOPF problem in which corrective actions are not considered in the system. This does not include the re-dispatch of generation units[4] along with the automatic response of adjustment elements such as the derivation and tap elements in transformers. Formulation including only preventive action includes equations 5-1a - 5-1e [44, 49].
- **Corrective Security Constrained Optimal Power Flow (CSCOPF)**: This formula considers generation re-dispacth for the removal of violations in the system. Formulation including corrective actions includes equations 2-3a - 2-1h [43, 47, 50, 51].

Different strategies have been used to reduce the solution time of these problems, as they require the evaluation of different contingencies. Some include serial and successive assessments of contingencies; however, the execution time is prolonged. Other approaches include the decomposing of the problem into one master problem and another group of subproblems (e.g.. Benders, Dantzig-Wolfe, Talukdar-Giras, etc.). Equations 2-2a - 2-3c, shows the approximation using such strategies [48]. The Equation 2-2a - 2-2c shows the master problem, while 2-3a - 2-3c is the result of subproblems when including security constraints.

$$\min \quad f_0(x_0, u_0) \tag{2-2a}$$

$$\text{s.t.} \quad g_0(x_0, u_0) = 0, \tag{2-2b}$$

$$h_0(x_0, u_0) \leq h^{max} \tag{2-2c}$$

The c contingency subproblems are then

$$\min \quad \underline{1}^T \cdot \epsilon_k \tag{2-3a}$$

$$\text{s.t.} \quad g_k(x_k^0, u_0 + \epsilon_k) = 0, \tag{2-3b}$$

$$h_k(x_k^0, u_0 + \epsilon_k) \leq h^{max} \tag{2-3c}$$

[4] Only automatic response of droop is considered in generators

The 1^T vector is a vector of ones, while ϵ_k represents the required preventive controls. This representation removes the readjustments, passing them as constraints to the master problem. In order to remove the violation the constraint is added as a Bender cut of the shape,where λ is the Lagrange multiplier vector associated with constraints.

$$1^T \cdot \epsilon_k + \lambda(u_0^* - u_0) \leq 0 \tag{2-4}$$

Similar methods [42, 46, 51] allow to include and parallelize the various systems, however, their application has been used in small networks of few nodes [42], with a limited number of contingencies and without having the advantages of vectorization achieved through existing computational structures [41, 52, 53].

2.2 Parallel and Heterogeneous Computer Architectures

High Performance Computing (HPC) and Parallel and Distributed Computing (PDC) are considered inseparable concepts nowadays [15, 54], [7, Introduction Paragraph 2]. This is the result of the interaction of different hardware structures that composed modern computers. Computers for multiple processing tasks include heterogeneous, parallel and distributed architectures [55, page 179]. Traditionally, HPC had been generally related with computers with high technical features that overpass normal computer specifications or common system performance [56]. E.g. Vector and Reduce Instruction Set Computer (RISC) processors were considered sophisticated in the past but nowadays may be taken for granted.

During this research, the concept of HPC will be related with Parallel Computing (PC) and Heterogeneous Computing (HC) concepts, that may differ with the conventional approach as noted by [56]. Definition of HPC that includes high performance computers is not pondered since computer technologies tend to change in the near term. E.g. Quatum Computing (QC) is a present trend that will change the existing terms of high performance computers. To avoid any possible confusion, the definition from [57] for Heterogeneous Computing (HC) will be adopted in this research while parallel processing will relate all structures that executes multiple tasks simultaneously. HC includes processors of different types, such as acrfullcpus, Graphical Processing Unit (GPU)s and Field Programmable Gate Array (FPGA)s. Nevertheless, this research is related with CPU + GPU environments.

In the following subsections, computer architecture for regular microarchitectures using CPUs and GPUs systems will be further discussed.

2.2.1 Modern Computer Architectures

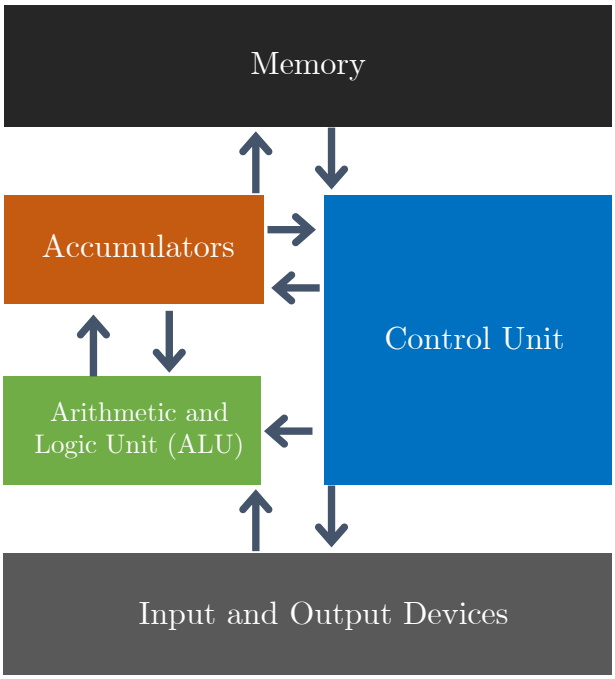

Figure 2-2: The canonical stored program architecture [2]

Modern architectures generally follow the canonical model for a stored program architecture viewed in Figure **2-2**. This generally consists of a Control Unit, at least one Arithmetic Logic Unit (ALU) and some kind of immediate memory (accumulators or registers) which allows for calculations. These are put together into a CPU which in turn interacts with longer term memories: main memories such as Random Access Memory (RAM) and main storage devices such as hard drives. More specifically, modern architectures add on this simpler scheme incorporating more advanced concepts and technologies such as Instruction Set Architecture (ISA), which specifies how binary instructions should be formatted and thus understood by the processor; main memories, which make it possible for an initial program to load other programs (e.g. an operating system); code branching, so a program can instruct what should be executed next; Reduce Instruction Set Computer (RISC) and Complex Instruction Set Computing (CISC) architectures, which make a compromise between hardware simplicity and compiler efficiency vs hardware complexity and coding efficiency; caches; super-scalar execution; parallelism; among others [2].

There are many reasons to include caches and intermediate memories into computing systems, one of the major ones being to take full advantage of temporal locality and spatial locality to increase performance. Locality generally refers to a common occurrence in

programs where recently (or close) accessed locations will be accessed again (or for the first time in the case of close addresses) [2]. However, it is important to note the differences in access speed and size as they relate to the different memory resources available. Memory access to a RAM resource typically takes much longer than accessing data in processor registers and caches, but in turn registers and caches have a higher price/storage ratio and hence typically have much less storage capacities. Such differences in speed, storage and price can be seen in Figure **2-3**, where the different components of a modern computer memory hierarchy are represented from top to bottom as they relate to the CPU, and Table **2-1**, where the general magnitudes of relevant properties for the memory hierarchy components are displayed.

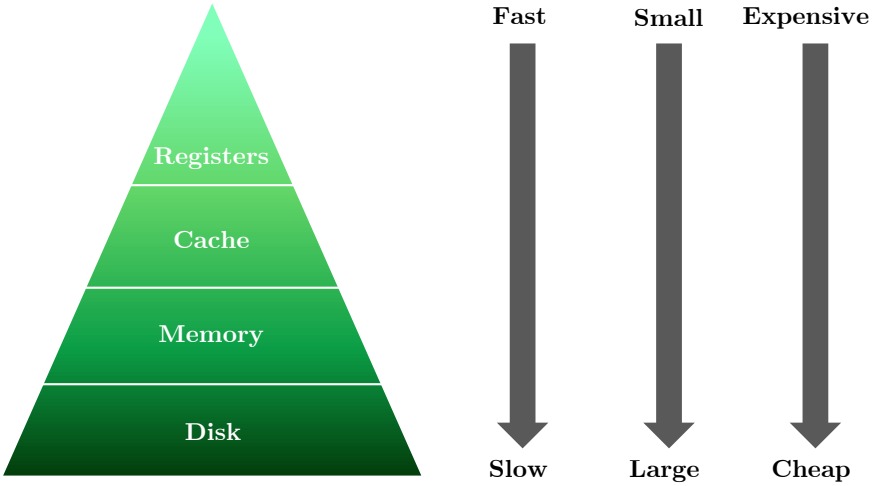

Figure 2-3: Typical memory hierarchy and performance for a modern computing system [2]

Table 2-1: Summary of memory hierarchy - Technical and economical specifications [2]

Level	1	2	3	4
Name	Registers	Cache	Memory	Disk
Size	˜1kB	˜1MB	˜1GB	˜1GB
Access time	˜1ns	˜10ns	˜100ns	˜1000ns
Bandwidth	˜10000MB/s	˜1000MB/s	˜100MB/s	˜10MB/s
Cost	˜100$/MB	˜100$/MB	˜10$/MB	˜0.1$/MB
Managed By	Programmer	Processor	OS	OS

As memory accesses are generally needed to retrieve instructions and data, the processor is normally able to operate only as fast as the memory, which is a problem known as the Von Neumann Bottleneck or Memory Wall [2]. Technologies such as the caches as mentioned

earlier help to mitigate this kind of problem.

A better understanding of these architectures can provide researchers with better tools for taking advantage of performance potential in terms of Instruction Level Parallelism (ILP), Pipelining and Heterogeneous Computing in general. Typically, parallelism in general purpose microarchitecture has been historically achieved through pipelining, which is basically dividing any process into sequential separate and shorter stages (adding associated short-term memory registers) in order to let data flow (and be processed) into a new stage as soon as the next stage has finished its previous data processing, thus allowing the throughput (output speed) to be as fast as the slowest processing stage. A more detailed explanation of pipelining can be found at [2].

2.2.2 GPUs

The first versions of microprocessors focused on the serial processing of [2, 58]. In this way, a new task was executed only after an ongoing process had finished. On the other hand, current processors have evolved in such a way that current CPUs integrate new structures that allow the parallelism of tasks, through structures and architectures that involve the operation of multiple cores and pipelining. However, [58] mentions that parallelism in multiple cores not only integrates the execution of arithmetic operations and direct parallelism of activities, but also includes complex tasks such as cache management, instruction decoding, and branch prediction, among others. On the other hand, the existence of vector structures in the GPU may allow better results using another structure for memory management and the execution of tasks. These vector processors allow the parallel execution of homogeneous processes [2]. The previous advances resulted from developments of hardware from manufacturers (NVIDIA and AMD) mainly focused on image processing; particularly, video games. These have made vector structure processing products, such as GPUs, available to the public [58]. Similarly, manufacturers have created languages like CUDA that allow for more GPU-friendly interaction.

As with most PHC systems, *it is almost always necessary to tune algorithm and software implementations in General Purpose Graphics Processing Unit (GPGPU)*[5] *for them to fit the parallel models, which requires an understanding of both architectural and programming models* [59] and [3, page 21].

[5]The terms GPGPU and GPU will be used indistinctly through this research. However, Computer Science terminology relates GPU with graphical processing units dedicated to graphics rendering. On the other hand, GPGPU is a GPU that performs specialized calculations that are usually performed by the CPU

2.2.2.1 GPU Microarchitecture

One way to exploit data parallelism is through the use of powerful co-processors as GPUs. This section presents an overview of the key aspects of GPU architecture, focusing particularly on Nvidia architectures.

Today's GPU architectures contain thousands of computer cores along with information flow management units. In this type of solution, several of the hardware components are in turn part of its hierarchical structure at the software level, such is the case of memory banks and caches. These elements are made up of several Streaming Multiprocessors (SM), which contain a scheduler (more generally a thread scheduler) that normally manages the flow of information packets called warps. This scheduler distributes arithmetic operations or processes through multiple processing units called Scalar Processor (SP). SPs can be integer SPs or floating SPs, which allow operations with integer or floating values depending on the precision and processing speed required. Turing architecture's SMs, as in previous Nvidia architectures, also comprise a plurality load/store units Load/Store units (LD/ST)s which calculate source and destination addresses and a plurality of Special Function Units Special Function Units (SFU) which accelerate the execution of transcendental instructions at hardware-level [3, 58, 59]. Figure **2-4** presents a schematic of the SM for Nvidia Turing GPU architectures [3], with many of the above-highlighted components.
Recently introduced by previous Nvidia architectures, Turing architectures include a plurality of TC, which are specialized execution units designed specifically for performing the tensor/matrix operations which in the context of GPGPU are specially useful for Deep Learning and Neural Networks applications [3]; also Turing includes Real-Time Ray Tracing (RT) Cores, which have applications mainly in tasks related to the rendering of visually realistic 3D models, although its applications on GPGPU may be worthwhile exploring in the future.

The HC GPGPU structure, includes two kind of devices: The "host" or CPU system and "device" or GPU system. The first is able to launch different processing task in the latter. The data flow allows copy and paste operations from and to the device's global memory (normally the RAM in the graphic card). In GPGPUs, there is also a threat-scheduler that distributes blocks of threads to different SMs for the final execution at SPs. Additionally, the L2 cache allows data caching for redundant global memory accesses made by SMs [59].

The SM register file provides registers for threads execution. The L1 cache allows additional data caching for global memory accesses made by different threads. Finally, shared memory allows different threads to explicitly share data [59].

Based on the previous computing structure, the use of appropriate software design that

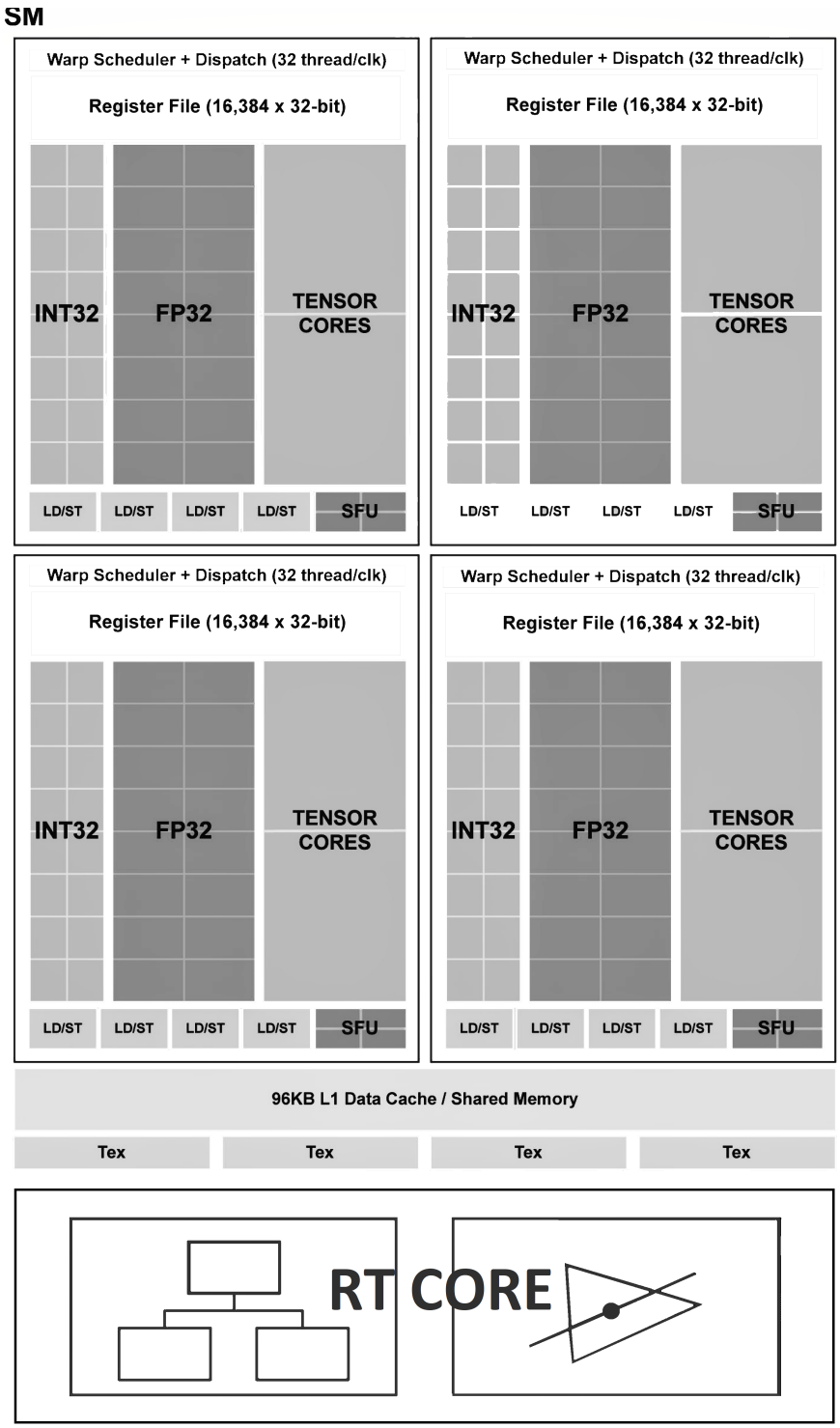

Figure 2-4: Turing TU102/TU104/TU106 Streaming Multiprocessor (SM) [3]

inter-operates efficiently with existing hardware is an inherent factor to speed up computing processing and satisfy real-time requirements.

2.3 Applications of PHC to Power Systems

The evolution of energy systems and the integration of new technologies, such as non-conventional renewables, distributed generation, battery energy storage systems, High Voltage Direct Current (HVDC) lines, among others, have made the simulation of the electrical system a challenge for the application of various services. In particular, larger and more complex systems require longer processing times to calculate and solve these problems. As a result, multiple authors have looked for alternatives to speed up the calculation time, in such a way that more optimal solutions are found in reduced computing ranges. One of these alternatives is the use of PHC to execute the tasks in a parallel way, avoiding the delays that sequential execution brings with it. In the last three decades, different developments have allowed the application of PHC architectures to power systems. The initial alternative was the use of PC clusters communicated through the use of Message Passing Interface (MPI). Subsequently, fog and cloud computing technologies were used not only to speed up calculation time, but also to manage large volumes of information, e.g. the data records in Smart grids. Finally, during the last decade, different advances suggested the use of FPGA and GPU architectures as useful and optimized mechanisms for the parallelization of tasks. The FPGA architecture allows applying techniques focused on Multiple Instruction, Multiple Data (MIMD), which enable the execution of different instructions at multiple points simultaneously. On the other hand, the GPU architecture is based on the Single Instruction, Multiple Data (SIMD) architecture that allows for simultaneous execution of the same task at multiple points. Then, technologies like fog and cloud computing were used not only to reduce simulation time, but also as a solution to handle all the measured data from smart grids. Finally, over the last decade, researchers have found in FPGA and GPU architectures a useful and optimized mechanism for parallelizing tasks. The FPGA architecture allows the application of the MIMD technique where different instructions can be simultaneously executed at multiple data points. On the other hand, the GPU architecture provides the ability to use the SIMD technique for common tasks where the same operation is performed on multiple data points at the same time. As a result of the GPU architecture, focused on SIMD operations and the various developments aligned with the programming of GPU instructions in non-graphic programming languages such as C, C ++, FORTRAN, among others, researchers have observed how to use GPUs for the solution of power system problems with structures presenting common recurring instructions. The literature review revealed more than 200 investigations that make use of PHC in the last three decades. Figure **2-5** shows studies of power systems using GPUs. Figure **2-6** shows power system

applications, using structures like PC clusters, fog, cloud computing and FPGAs.

In Figure **2-5**, it is shown that Power Flow Analysis is the application with the most research using GPU. Different authors have focused their projects on reducing the convergence time of the methods for calculating the solution of flow equations. This has been achieved through the parallelization of specific stages of the iterative algorithms using the GPU. Depending on the algorithm and the type of system (Transmission or Distribution), different strategies have allowed the parallelization of steps such as impedance and Jacobian matrix construction for solving power flow problems using GPUs. In [60], a strategy is proposed to accelerate the convergence rate of the gradient algorithm conjugated with a multigrid preconditioning method. The strategy is implemented in GPU accelerating the traditional analysis using the Direct Current (DC) algorithm. In [61], the integration of a GPU solver is used to improve the convergence rate of a conjugate gradient algorithm using a Chebyshev polynomial preconditioner. In [62], the authors make a proposal for a preconditioned conjugate gradient solver that uses the mixture of two alternatives: preconditioner Jacobi diagonal and polynomial Chebyshev to solve the load flow problem through an algorithm that integrates the Fast Decouple Power Flow algorithm integrated with the Inexact Newton on GPU. In [63], efficient data management strategies are used through parallelism with primitive parallel functions such as map function, reduction, and scan for various steps of the algorithm. The strategies look for acceleration of the Newton-Raphson and Gaussian power flow algorithms using GPU.

The second GPU application area with the most citations was Transient Stability (TS) and Electromagnetic Transient (EMT). In [64] and [65], the authors propose the parallelization of the non-linear Differential-Algebraic Equations (DAEs), since they present discretized and vectorized structures. This allows the slow coherence method to be applied to simultaneously divide TS calculations into multiple processors. The structure is appropriate to use a SIMD strategy and solve the Differential-Algebraic Equations (DAEs) system. In such a way that the authors show a hybrid CPU-GPU to solve dispersed large matrices in parallel. On the other hand, the EMT area shows relevant GPU applications. In [66] the models of electrical elements such as lines, motors, and generators are represented through discretized models (lumped models, Universal Line Models (ULM) and Unified Machine Models (UMM)). These models offer a suitable structure in terms of vectorization and linearization and can be solved on the GPU. In [67] hybrid algorithms for homogeneous and heterogeneous computation are proposed in order to solve large systems. In heterogeneous computing, control signals and Norton equivalent current sources representing different electrical elements are processed. The nodal injection currents are solved on the GPU, using a Layered Directed Acyclic Graph (LDAG) method. The method sequentially links parallel primitives and fused multiply-add that performs add operation for the linear models. Finally, the nodal voltages are calculated from the nodal injection

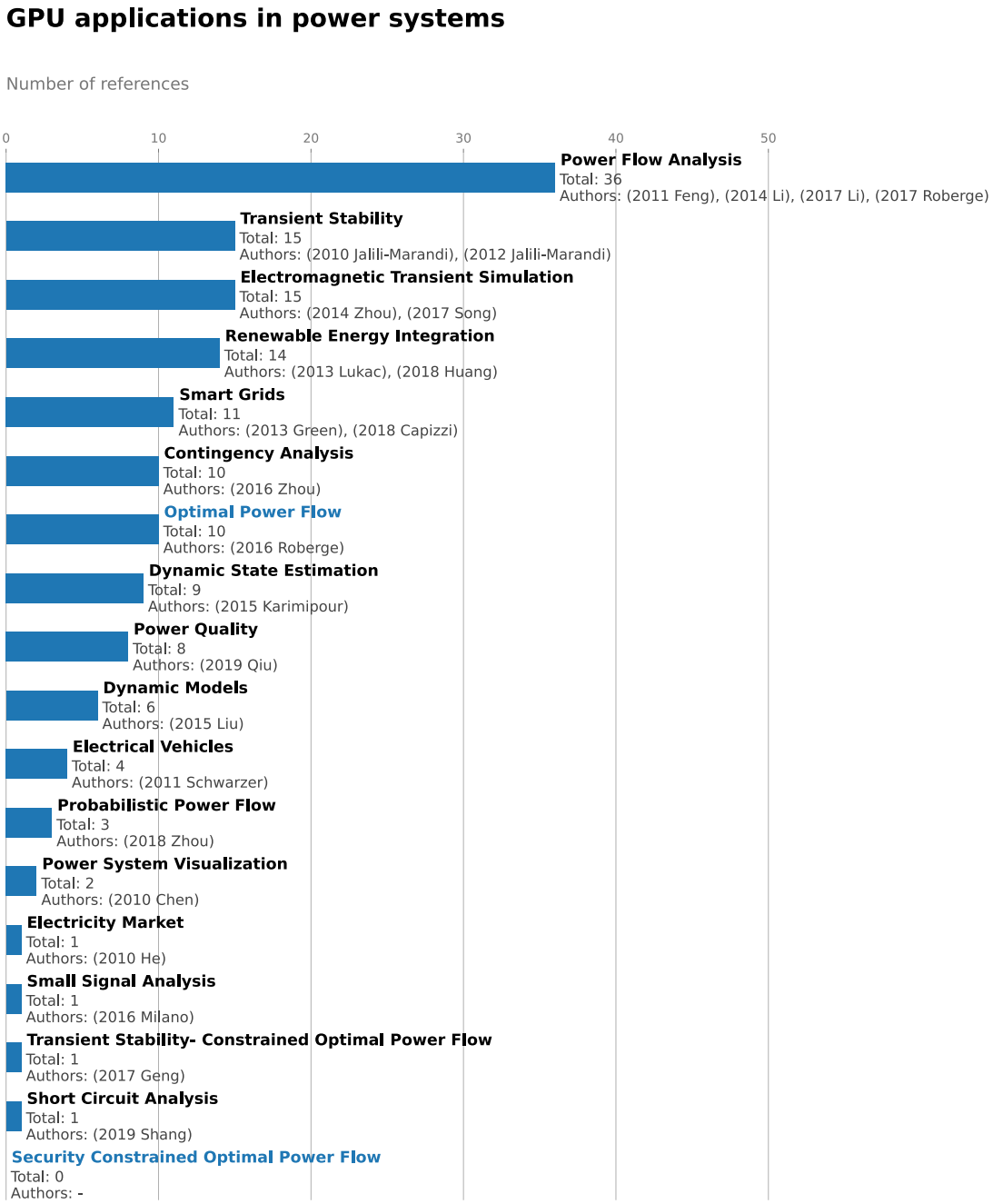

Figure 2-5: GPU applications in power systems

currents and the existing impedance matrix.

The following applications with the greatest use are smart grids and renewable integration, which largely integrate the use of GPUs. GPU structures have allowed the integration of measurements of meteorological conditions and ranging data [68]. On the other hand, it has been applied to solve operational planning processes such as OPF that include the uncertainty of non-conventional renewable generation resources [41]. [69] performs a review of PHC applications to power systems. In this review, the GPU is shown as a suitable alternative for the visualization of smart grids in real time and in off-line calculations. Finally, in [70] a distribution in a GPU cloud system is proposed to solve a dispatch problem in real time. In that study, the use of a recurrent neural network wavelet is proposed to perform generation dispatch.

The next with a large number of citations in GPU use applied to power systems is the contingency analysis and OPF. In these approximations the OPF is used to reduce simulation times in the energy flow analysis. Since the algorithms can perform a greater number of iterations with shorter times, it is expected to find better optimals in shorter time frames. The strategies propose hybrid CPU-GPU structures to solve a large number of contingencies and scenarios. [71] proposes a contingency screening using Decoupled Power Flow (DEC) using GPU in order to speed up the contingency ranking. The algorithm includes calculations of nodal voltages and calculations of power flows in lines during N-1 conditions of lines and generators. Regarding OPF application, Section 2.4 presents how OPF is parallelized using the SIMD technique.

After the application of optimization in the operation of GPU-OPF networks, estimation of dynamic state and power quality are listed as relevant in the use of GPU. The dynamic estimation of the power system state is implemented with an extended method of a two-level Kalman filter, in which a CPU-GPU structure is used [72]. The stages of parameter identification, prediction, and state filtering of the Kalman filter are executed in parallel on GPU. In the same way, the calculation of the Jacobian matrix and the system of linear equations solution for the estimation of system states is also executed in GPU. Regarding the area of power quality, [73] shows an optimized algorithm that uses the modified S transform and a parallel stacked sparse auto-encode to capture disturbances in current and voltage wave forms. Another field of use of GPU is the acceleration of algorithms for the identification of variables during the processes of model validation. [74] uses GPU-accelerated Particle Swarm Optimization (PSO) algorithms to identify parameters in models of synchronous machines such as permanent magnets.

Other references that use GPU architectures (less than 5 citations) include: Optimization for the definition of design variables of electric vehicles [75], reduction of the calculation

time of Monte Carlo simulations with simple random sampling in order to execute probabilistic power flow calculations [76], visualization of electrical networks in real time with elevation models for friendly interaction with electrical variables [77], energy demand forecasting using computational intelligence techniques such as Levenberg-Marquardt learning through a multilayer perceptron architecture of a neural network [78], approximation of small signal solutions of large networks using different solution methods: Chebyshev discretization, time integration operator discretization, linearmultistep and Pad e approximant [79] and acceleration of the short-circuit current contributions through various branches when there is a fault at specific points. In the latter, the authors use the GPU to accelerate the inversion of the admittance matrix to later perform the calculation of injection currents at various points with the SIMD technique [80]. Regarding Transient Stability-Constrained Optimal Power Flow (TSCOPF), section 2.4 presents a hybrid CPU-GPU approach to accelerate the constraint analysis of each contingency and validate the TS of the optimal solution. For the SCOPF application, no previous work was found.

On the other hand, Figure **2-6** describes the relevant contributions in power system applications using PC clusters, fog and parallel computing, and FPGAs. These strategies do not follow the SIMD architecture but rather use another structure for the parallelization of activities. In this type of structure, the most relevant activity is TS analysis. In [81] a proposal is given using the relaxed and very dishonest Newton's method to solve discretized algebraic equations using the trapezoidal rule. The authors in this study show the creation of a cluster that integrates high-performance processing on an InteliPSC / 2 supercomputer and the shared memory of an Alliant FX / 8 system. In [82], parallelization is performed by dividing subsystems using the Interlaced Alternating Implicit (IAI) algorithm. The algorithm parallelizes in subsystems and hierarchically solves each system of DAEs independently using a block bordered diagonal form algorithm. In [83], dynamic simulation results of large networks are shown using a decomposition in subsystems through a shared model of parallel programming. The method divides the large network into smaller networks and solves them independently using multi-core computers and OpenMP.

The next area of application for these PHC strategies is contingency analysis. In [84] and [85] an asynchronous strategy is shown that allows optimization of available resources to reduce the execution time of thousands of contingencies in large-scale networks. This alternative comprises a master-slave structure. The master schedules the clients and each client executes the contingencies. Also this proposal uses scheduling and stealing methodologies to optimize the load balancing of workers (slaves). This allows all workers to always be executing tasks, in such a way that as soon as a client is available, it is assigned with new contingencies in order to optimize the use of resources. Master-slave communication is achieved via MPI.

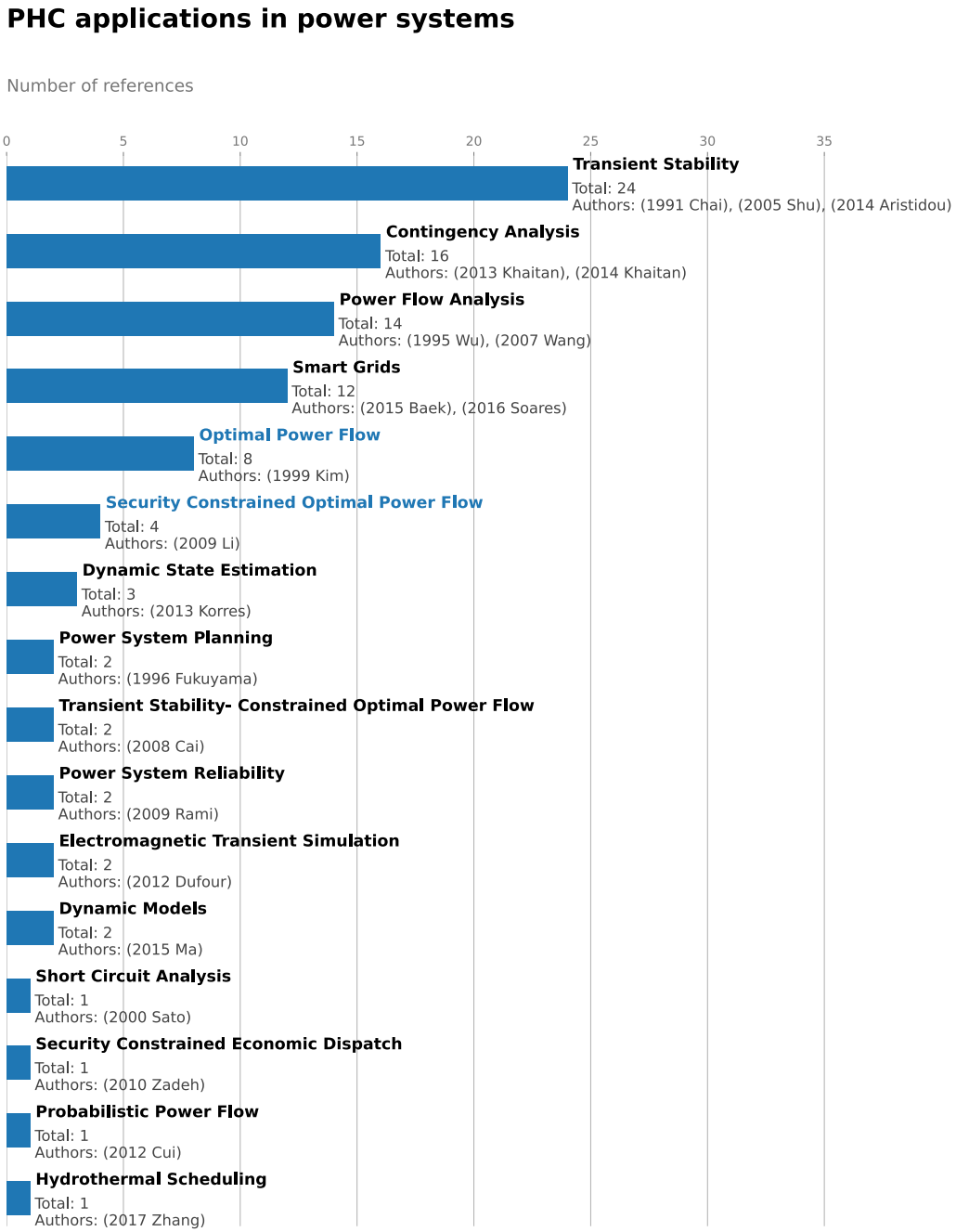

Figure 2-6: PHC applications in power systems

After contingency analysis, power flow analysis and smart grid applications are most frequently cited in PHC architectures. In [86], a parallel LU decomposition algorithm is implemented using 20 multi-core processors in order to solve systems of equations represented in large matrices. Factorization and parallelization is applied with Newton and Fast Decoupled algorithms in order to reduce execution times. [87] presents a parallelization of the Jacobian matrix that allows parallelization of the LU factorization. The algorithm is implemented in a computer that also contains an FPGA. The smart grids area shows applications for the control and operation of networks in real time. [88] presents a cloud structure that allows for big data processing, in such a way that scalability, agility and flexibility are guaranteed. Similarly, the authors highlight the need to guarantee cybersecurity based on the relationship of identity, signature and proxy encryption. In [89], a generation programming proposal is made. It integrates the management of distributed resources together with electric vehicles to satisfy the energy demand. The problem is solved through PSO algorithms and a Mixed Integer Linear programming technique. The objective functions include the costs of: distributed generation, demand response, charging and discharging of Battery Energy Storage Systems (BESS), charging of electric vehicles, non-supplied demand, generation curtailment and energy from external networks. Each multi-objective problem or combination is solved independently in each computer core.

The next applications with more usage of PHC technologies are OPF and SCOPF with eight and four previous works, respectively. Section 2.4 shows parallel approaches of OPF and SCOPF where the optimization problem is decomposed in reduced subproblems that are solved in independent processor units. The solutions are implemented in CPU clusters.

The latest application group of PHC technologies in power grids includes (less than four benchmarks): dynamic state estimation, power network planning, TSCOPF, reliability, EMT studies, dynamic modeling, short circuit analysis, simple economic dispatch with safety restrictions in generation, probabilistic load flow for hydrothermal generation. [90] shows the parallelization of a large power system where the state estimator is divided into each area individually. The areas exchange information through a central coordinator. The alternative identifies areas of similar sizes in order to homogeneously divide the computational loads across the existing computer network. The MPI high-performance communication (MPICH2) allows the interaction of the coordinator and the various areas.

[91] performs a multi-period generation scheduling. The method integrates a cluster of transputers. The coarse-grain version of the parallel genetic algorithm is implemented. The subpopulations are distributed in different processes, and information is exchanged among them. An approach of TSCOPF implemented in a PC-cluster is shown in section 2.4.

In [92] the authors propose a parallel metaheuristic method in order to solve a TSCOPF

problem that seeks to identify the optimal reinforcements of the network in order to guarantee the desired reliability in a power network. [93] analyzes and compares real-time TS solution algorithms and instantaneous relaxation algorithms. On the other hand, nodal variations are also compared with EMT simulations in real time. The computing system includes multi-core, multiprocessor, and FPGA computers. [94] implements a PSO algorithm in Open CL to identify the parameters of photovoltaic models. The evaluation of the different particles is carried out in parallel simultaneously.

In [95] probability density curves are created by parallelizing the execution of Monte Carlo simulations in different virtual machines. [96] performs an economic dispatch using an Interior Point Optimization (IPOPT) strategy including generation limits, system losses, and chunk generation cost functions. The algorithm validates that the limits in lines are met, guaranteeing the security of the system. [97] performs a probabilistic power flow to assess generation uncertainty by dividing Monte Carlo routines across multiple CPU cores. Finally, [98] includes a differential evolution algorithm for optimizing the dispatch of a hydrothermal generation unit including power flow restrictions. Multiple populations are divided into individual processors in such a way that there are independent solutions.

2.4 Applications of PHC to OPF and SCOPF

As shown in 2.3, different applications of the SIMD architecture in GPU have been focused on OPF problems, including TSCOPF. These problems have been solved through hybrid strategies that include CPU-GPU elements. Figure **2-7** shows an example of a hybrid CPU-GPU OPF algorithm based on a Metaheuristic Method (MM) and the Newton-Raphson algorithm. The MM is used to find the optimal operating variables of the electrical system using voltages in buses and generation from the machines, so that the greatest benefit and the lowest cost are obtained. The algorithm shows the tasks performed sequentially in CPU and parallelly in GPU. CPU-GPU and GPU-CPU data migration seeks to be minimized in order to avoid bottlenecks associated with long transfer times. This last activity is one of the main challenges of the CPU-GPU interaction.

The algorithm starts loading the power system data, then the selected MM is initialized. After the initialization, the MM instructions for the first iteration are performed. The CPU transfers the power system data to the GPU allocating the required memory. The first task executed by the GPU corresponds to computing the bus voltages based on voltage magnitudes and angles. Then, GPU calculates the Ybus. The CPU transfers the Ybus from the GPU. To properly allocate the required GPU memory to store the Jacobian matrix, the CPU computes the non-zero elements of the Jacobian matrix based on the Ybus information. Then, the CPU transfers NNZ to the GPU allocating the required

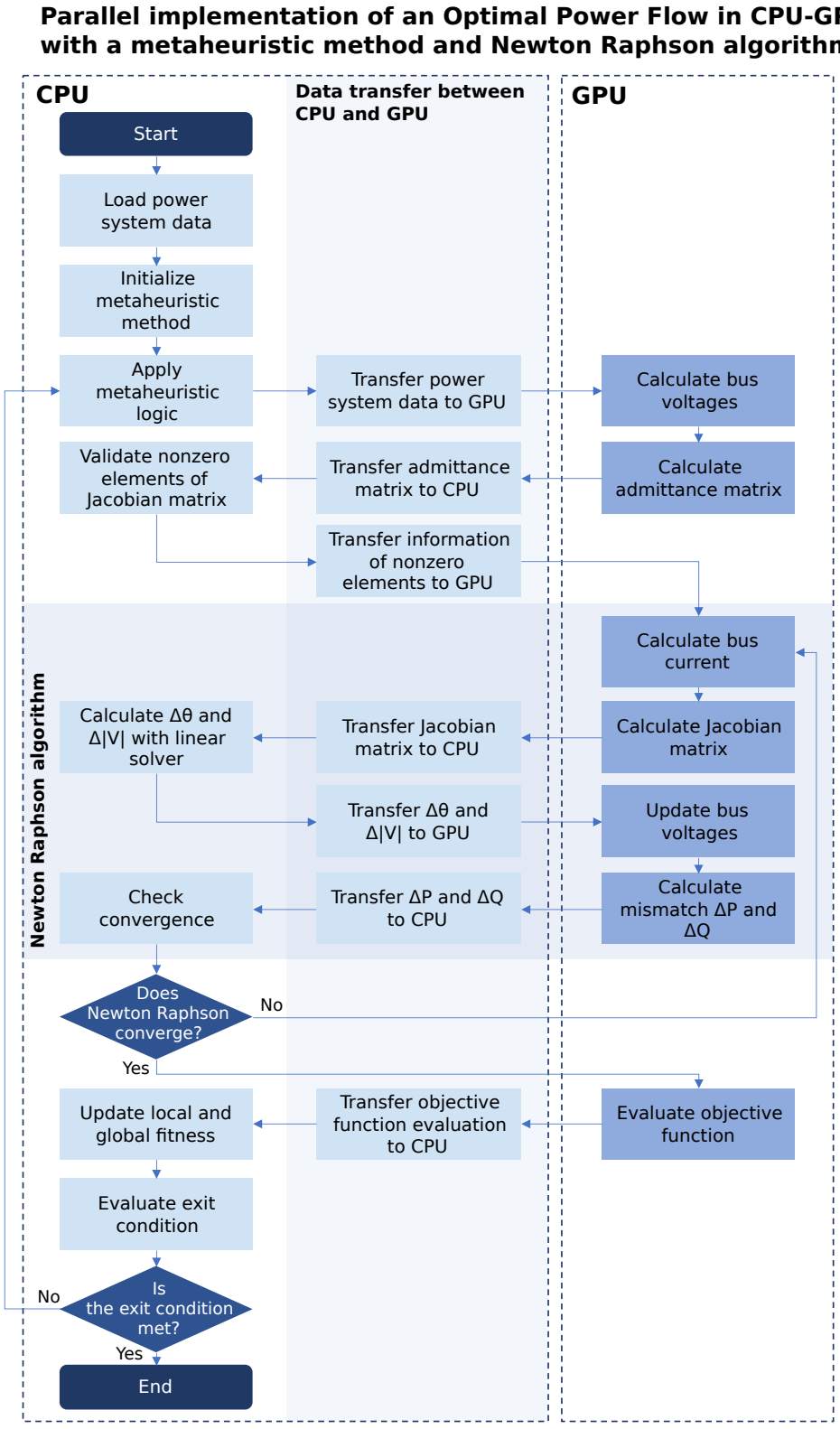

Figure 2-7: Parallel OPF algorithm using CPU and GPU platforms

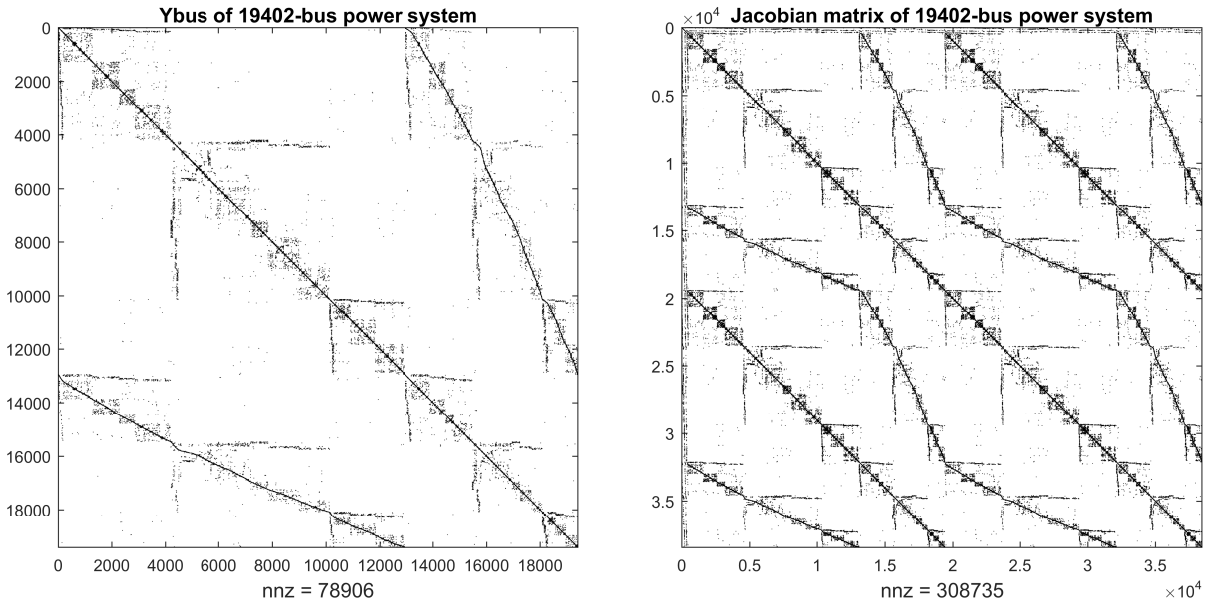

Figure 2-8: Ybus and Jacobian matrices of 19402-bus power system

memory.

For the first iteration of the Newton-Raphson algorithm, the GPU calculates the bus currents based on the bus voltages and the Ybus that are already in the GPU memory. Once bus currents are calculated, the GPU computes the Jacobian matrix. The CPU transfers the Jacobian matrix from the GPU. Then, the CPU calculates the power flow unknown variables using a linear solver and transfers the solution of the system of linear equations to the GPU. Then, the GPU updates the bus voltages and computes the mismatch equations. Finalizing the Newton-Raphson algorithm, the CPU transfers the mismatch equations from the GPU and checks the algorithm convergence. If Newton-Raphson converged, the OPF follows to the GPU for evaluating the defined objective function. Otherwise, the algorithm returns to the bus current calculation in the GPU.

Once the objective function is evaluated for all elements of the MM, the CPU transfers the objective function results from the GPU. Finally, the CPU updates the local fitness for each MM element; similarly, the global fitness with the best local fitness. The CPU evaluates the defined exit condition for the MM. If the exit condition is met, the MM ends returning the optimal solution found during the OPF execution. Otherwise, the algorithm returns to the task where the MM instructions for the next iteration are performed.

[13] and [99] use a parallelized optimal power flow structure as described in Figure **2-7**. The authors of these papers include a solution to the OPF problem using MM and Newton

Raphson algorithms on an HPC platform that includes interaction CPU-GPU. The PSO algorithm is used to find optimal operating points that consider generation cost, transmission losses, and environmental costs related to large-scale CO^2 emissions. In this research, the authors use the GPU to parallelize the calculation of the different iterations that initialize the particles' position and velocity, the fitness of all the particles, the update of the position, and the movement of the particles and the swarm.

Regarding TSCOPF, [100] performs the parallelization of an algorithm that includes optimization of a power system and the calculation of transient stability in different generation scenarios. The algorithm runs OPFs on the CPU and does a transient stability calculation at each OPF iteration using a GPU platform. The review did not find a PHC application that includes SCOPF through CPU-GPU algorithms that will take the potential of each hardware element.

On the other hand, Section 2.3 shows the usages in OPF and SCOPF applications of not only GPU architecture but also PHC technologies such as PC-clusters, fog and cloud computing, and FPGA. [101] presents a parallel OPF solution developed on a computer cluster. The system is divided into different geographic regions that optimize the dispatch of units in each area. At each iteration the voltage vectors are updated in magnitude and angle. To interconnect the areas in the network, nodes are used with fictitious generators that inject or consume active and reactive power and exchange information with the other areas through interconnection interfaces. The OPF uses an IPOPT method. System tests were performed on Sun Ultra Sparc workstations.

In [102] a parallel DE algorithm is presented to solve a TSCOPF problem. The algorithm uses Point of Wave (POW) and Transient Energy Function (TEF) simulations. The optimization function includes the costs associated with generation in the system. The algorithm evaluates the security of each generation dispatch through simulations in steady state and faulted conditions. Initially the rotor angles are calculated and later the TEF to define the expected stability in the system. The algorithm is implemented in a Beowulf PC cluster with a machine acting as a master and another 30 worker nodes that execute operations as slaves. The communication is executed with MPI protocols. The general population is divided into subpopulations so that they execute the DE algorithm at each node. Each run calculates the load flow, fitness, and TS. The control node manages the initialization, reproduction, and updating of the individuals in the various populations.

In [48] the decomposition of a SCOPF problem is carried out using a Bender decomposition algorithm. The decomposition includes a master problem, along with slave nodes that solve the algorithm and add breaks at each iteration. The problem only includes a preventive SCOPF that a master OPF problem and N subproblems that verify correct operation in

the event of contingency N contingencies. The problem solves the base OPF problem and independently dispatches the power networks with each contingency included. If the system with the contingency applied is feasible, the base solution is accepted, otherwise a Bender cut or constraint is added to the master problem. The algorithm is executed until the necessary cuts are added in such a way as to guarantee an optimal and secure solution.

3 Parallel Power Flow Algorithm in Low Cost Embedded Computer Architectures Empowered by GPU

The execution of power flows in real time as well as supervision and control systems such as SCADA and EMS have increased their importance significantly today. This has been the result of a greater number of devices that generate and consume energy throughout the day, such as energy storage systems, demand response and distributed generation.

In this chapter, a low-cost alternative for the rapid execution of power flows is described. These strategies are relevant to guarantee the security of the different elements existing in the network. By means of power flows that determine the voltage and current values, the high and low voltage values are determined at different points of the network as substations; overloads in transformers, lines and other devices; required levels of generation at different points in the system, etc. Since these values have been highly relevant, they have been executed on computers with high performance characteristics, mainly desktop computers and also Personal Computers (PEC). These specifications turn out to be expensive when it is required to process systems with a high number of buses, lines, generators and loads. This fact, together with the cost of software licensing, has considerably increased the costs of these solutions. In systems as microgrids, this cost is considerable in on-grid and islanded systems. [103, 104]

To reduce these costs in transmission and distribution systems, as well as small-scale energy systems as microgrids, low-cost systems such as embedded computers have appeared, which allow the execution of tasks of high technical requirement with vectorized structures. [105] shows a normalized cost of different microgrids depending on size and microgrid's application (Campus/Institutional, Commercial/Industrial,Community and Utility Scale). In the report, authors show that software and hardware for control plays a relevant role in microgrids cost of small size (7-10 kW) systems. A PEC cost is 10 to 20 times higher than ECs. An additional feature for ECs is their energy efficiency that requires less energy consumption to process complex tasks [106]. Those features makes ECs a promising alternative to increase microgrids penetration without penalizing entire system performance.

On the other hand, ECs can include a GPU for processing tasks with multiple processor to accelerate certain repetitive routines that can run with parallel structures as described in Section 3.2. Previous works had been performed in power system areas as steady state and transient states [107], power flow, OPF and metaheuristic optimization [13, 41, 108], etc. The approaches showed time speed up increase with CPU-GPU structures.

All previous approaches had integrated CPU-GPU structures; however, their implementation had been used in hardware based on PEC or desktops that include GPUs. Commonly GPU integrated in ECs are not as powerful as the prior structures. The reason is that execution time is highly dependent of hardware specifications and tasks requirements. In the case of microgrids with multiple measurements and large power system with time responses in less than 1 minute. The solution is considerable feasible and properly fits the time window for fast screening during on-line screening or operation planning. E.g. microgrids can use power flow simulation for early system analysis to guarantee secure system operation or to include generation and load re-dispatches or curtailments. Similarly, ECs can be used for frequency and voltage control. In frequency regulation, primary and secondary control actions, start up or shut down of generation unit, voltage control, OPF are some of the functions included in network controllers that can be implemented in ECs platforms [109–111]. The platforms can interact with data concentrator units that capture and send control signals (circuit breaker states, current and voltage magnitudes, active and reactive power, etc.) from several Intelligent Electronic Devices (IED) through wired lines or wireless communication channels.

To keep power system security during normal and contingency operation online load flow analysis is performed using a vectorized algorithm to be implemented in a low cost EC wtih GPU atchitectured. The algorithm's structure uses a vectorized Newton-Raphson method to take advantage of the GPU architecture. The parallel structure as the entire algorithm and results are shown in the sections below.

3.1 Power Flow Architecture

During planning, operation and control of power systems load flow assessment is performed to guarantee that all electrical elements operate inside secure limits. In steady state, this analysis uses the node-voltage equations that are solved based on the kirchhoff's current law, network impedance, grid configuration, loads and generators in the system. For AC system,

the nodal equations are described as follows:

$$P_i = \sum_{k=1}^{N} |E_i||E_k| \left[G_{ik}\cos\left(\theta_i - \theta_k\right) + B_{ik}\sin\left(\theta_i - \theta_k\right)\right] \tag{3-1}$$

$$Q_i = \sum_{k=1}^{N} |E_i||E_k| \left[G_{ik}\sin\left(\theta_i - \theta_k\right) - B_{ik}\cos\left(\theta_i - \theta_k\right)\right] \tag{3-2}$$

Where:

N = Number of buses of the system
P_i = Active power of the i_{th} bus
Q_i = Reactive power of the i_{th} bus
E_i = Voltage magnitudes regarding the i_{th} bus
E_k = Voltage magnitudes regarding the k_{th} bus
θ_i = Voltage phase angles regarding the i_{th} bus
θ_k = Voltage phase angles regarding the k_{th} bus
G_{ik} = Conductance of the ij_{th} component of the Y_{bus} matrix
B_{ik} = Susceptance of the ij_{th} component of the Y_{bus} matrix

Given the non-linearity of these equations, the Newton Raphson method is well know method to solve this system of equations iteratively. To implement this method, it is necessary to compute the Jacobian matrix ($\mathbf{J}$) defined by:

$$\begin{bmatrix} \Delta P_2^{(k)} \\ \vdots \\ \Delta P_n^{(k)} \\ \hline \Delta Q_2^{(k)} \\ \vdots \\ \Delta Q_n^{(k)} \end{bmatrix} = \left[\begin{array}{ccc|ccc} \frac{\partial P_2}{\partial \delta_2}^{(k)} & \cdots & \frac{\partial P_2}{\partial \delta_n}^{(k)} & \frac{\partial P_2}{\partial |V_2|}^{(k)} & \cdots & \frac{\partial P_2}{\partial |V_n|}^{(k)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_n}{\partial \delta_2}^{(k)} & \cdots & \frac{\partial P_n}{\partial \delta_n}^{(k)} & \frac{\partial P_n}{\partial |V_2|}^{(k)} & \cdots & \frac{\partial P_n}{\partial |V_n|}^{(k)} \\ \hline \frac{\partial Q_2}{\partial \delta_2}^{(k)} & \cdots & \frac{\partial Q_2}{\partial \delta_n}^{(k)} & \frac{\partial Q_2}{\partial |V_2|}^{(k)} & \cdots & \frac{\partial Q_2}{\partial |V_n|}^{(k)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial \delta_2}^{(k)} & \cdots & \frac{\partial Q_n}{\partial \delta_n}^{(k)} & \frac{\partial Q_n}{\partial |V_2|}^{(k)} & \cdots & \frac{\partial Q_n}{\partial |V_n|}^{(k)} \end{array}\right] \begin{bmatrix} \Delta \delta_2^{(k)} \\ \vdots \\ \Delta \delta_n^{(k)} \\ \hline \Delta |V_2^{(k)}| \\ \vdots \\ \Delta |V_n^{(k)}| \end{bmatrix} \tag{3-3}$$

Equation 3-3 in short form is written as

$$\mathbf{J} = \begin{bmatrix} \frac{\partial P_i}{\partial \theta} & \frac{\partial P_i}{\partial |E|} \\ \frac{\partial Q_i}{\partial \theta} & \frac{\partial Q_i}{\partial |E|} \end{bmatrix} \tag{3-4}$$

The expressions to compute **J** are presented in Equations 3-5 to 3-8.

$$\frac{\partial P_i}{\partial \theta_k} = |E_i||E_k| \left[G_{ik} \sin(\theta_i - \theta_k) - B_{ik} \cos(\theta_i - \theta_k)\right] \tag{3-5}$$

$$\frac{\partial P_i}{\partial |E_k|} = |E_i| \left[G_{ik} \cos(\theta_i - \theta_k) + B_{ik} \sin(\theta_i - \theta_k)\right] \tag{3-6}$$

$$\frac{\partial Q_i}{\partial \theta_k} = -|E_i||E_k| \left[G_{ik} \cos(\theta_i - \theta_k) + B_{ik} \sin(\theta_i - \theta_k)\right] \tag{3-7}$$

$$\frac{\partial Q_i}{\partial |E_k|} = |E_i| \left[G_{ik} \sin(\theta_i - \theta_k) - B_{ik} \cos(\theta_i - \theta_k)\right] \tag{3-8}$$

The power residuals ΔP_i and ΔQ_i are the difference of the scheduled and calculated power values:

$$\Delta P_i = P_i^{sch} - P_i \tag{3-9}$$

$$\Delta Q_i = Q_i^{sch} - Q_i \tag{3-10}$$

The new estimates for bus voltage magnitudes and angles are:

$$\theta_{new} = \theta + \Delta\theta \tag{3-11}$$

$$\mid E_{new} \mid = \mid E \mid + \mid \Delta E \mid \tag{3-12}$$

Algorithm 1 shows the Newton Raphson algorithm where the inputs are the data of the system branches to build the Y_{bus} matrix, each generator's power ($\mathbf{P}_{gen}, \mathbf{Q}_{gen}$), and the information of the system loads ($\mathbf{P}_{load}, \mathbf{Q}_{load}$).

The algorithm outputs are the nodal voltage magnitudes and angles. In order to exploit the GPU potential in an EC sections in green are proposed to be vectorized as described in Section 3.2 and Section 3.3.

3.2 Power Flow Vectorization

The operation vectorization consists of modeling one or more tasks of an algorithm as vector or matrix operations. The first step is identifying the common operations within an algorithm. Generally, these common operations are known as single instructions. These instructions correspond to tasks where calculations for a data set are not related to calculations for another data set. The basic example of this type of instruction is the

Algorithm 1 NR Algorithm for LF Analysis

procedure LF_NR_EC(Branch,P_{gen},Q_{gen},P_{load},Q_{load})
  $\mathbf{E}, \theta \leftarrow [1], [0]$
  $\mathbf{Y_{bus}} \leftarrow YbusCalc\,(\mathbf{Branch})$
  $\mathbf{P_{net}} \leftarrow \mathbf{P_{gen}} - \mathbf{P_{load}}$
  $\mathbf{Q_{net}} \leftarrow \mathbf{Q_{gen}} - \mathbf{Q_{load}}$
  for $j \leftarrow 0$ **to** 10 **step** 1 **do** ▷ LF-NR - Iteration
    $\mathbf{P}, \mathbf{Q}, \mathbf{J} \leftarrow f\,(\mathbf{Y_{bus}}, \mathbf{E}, \theta)$ ▷ eqs. (3-1), (3-2) and (3-5) to (3-8)
    $\Delta\mathbf{P}, \Delta\mathbf{Q} \leftarrow \mathbf{P_{net}} - \mathbf{P}, \mathbf{Q_{net}} - \mathbf{Q}$
    $\epsilon \leftarrow max\,(abs\,(\Delta\mathbf{P}, \Delta\mathbf{Q}))$
    if $\epsilon < 1 \times 10^{-7}$ **then**
      Break
    end if

    $\begin{bmatrix}\Delta\theta \\ \Delta\mathbf{E}\end{bmatrix} \leftarrow \left[\mathbf{J}\right]^{-1} \begin{bmatrix}\Delta\mathbf{P} \\ \Delta\mathbf{Q}\end{bmatrix}$ ▷ eq. (3-4)

    $\theta, \mathbf{E} = \theta + \Delta\theta, \mathbf{E} + \Delta\mathbf{E}$
  end for
  return $\mathbf{E}, \theta$
end procedure

addition of vectors. The addition is performed element by element, and this operation only requires the information from the vector data. The addition of the set of elements of the vectors at the same position is independent of the operations applied to the remaining elements at the other positions.

Regarding the power flow algorithm, tasks such as computing the Y_{bus} matrix, calculating the Jacobian matrix, and updating the active and reactive power mismatch vector are common operations. These operations are single instructions that correspond to the active and reactive power balance equations and the first-order partial derivatives of these equations regarding the voltage magnitudes and angles.

Figures **3-1**, **3-2**, and **3-3** present the parallel calculation of the active and reactive power vectors for all power system buses. Figure **3-1** shows the single instruction to evaluate and the required data of the power system. As shown in Figure **3-1**, the necessary information to calculate the active and reactive powers for all buses are the voltage magnitudes and angles and the corresponding branch conductances and susceptances. This information is transferred from the CPU memory to the GPU memory. Once the information is in the GPU's global memory, the definition of the single instruction as a function is implemented

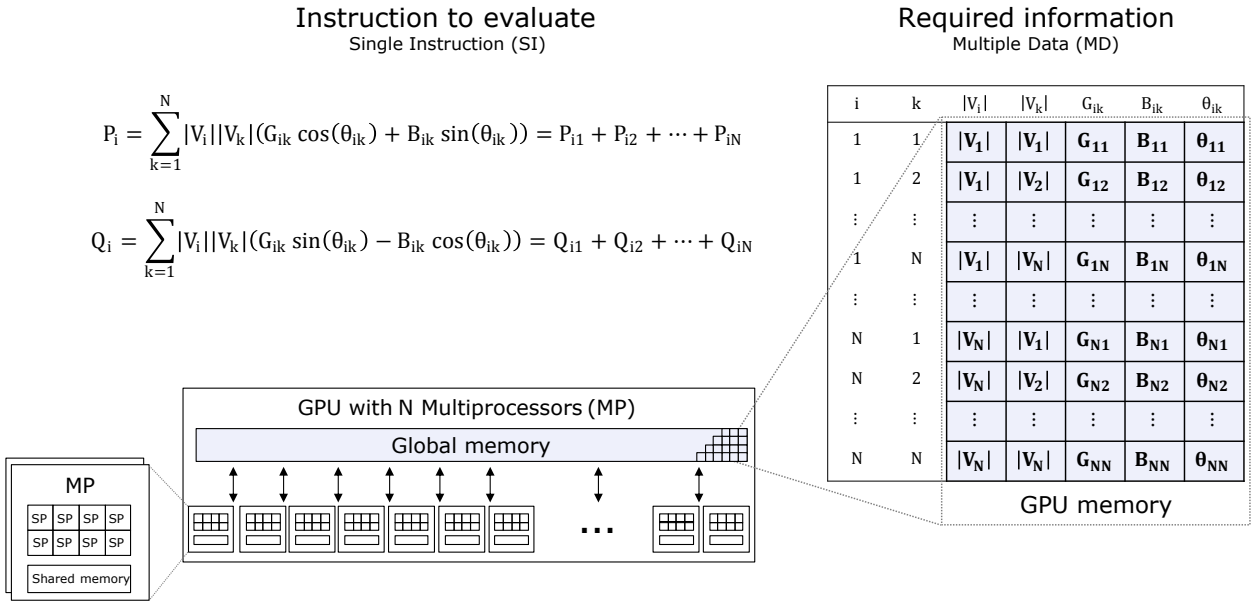

Figure 3-1: Definition of instruction to evaluate and the information required

to evaluate each P_{ik} and Q_{ik} parallelly in the GPU.

Figure **3-2** shows how the GPU calculates all P_{ik} and Q_{ik} parallelly. The GPU uses the information from its global memory and creates a kernel where defines the required resources (number of blocks and threads) to perform the instruction evaluation. The kernel creates a grid where the number of tasks running in parallel corresponds to the number of blocked times the number of threads. Based on the number of blocks and threads, the GPU thread scheduler assigns the multiprocessors and the corresponding scalar processors required to execute the calculation. Each scalar processor computes one of the P_{ik} and Q_{ik} equations. All scalar processors, depending on the GPU capabilities, execute the evaluation parallelly. Suppose the number of blocks and threads exceeds the limit of evaluations that the GPU can run simultaneously. In that case, the wrap scheduler creates a queue to execute all evaluations grid by grid.

Finally, Figure **3-3** shows how the GPU saves all P_{ik} and Q_{ik} values calculated in Figure **3-2**. Once all evaluations finish, the kernel saves the result in the GPU memory. This information is available for other kernel executions.

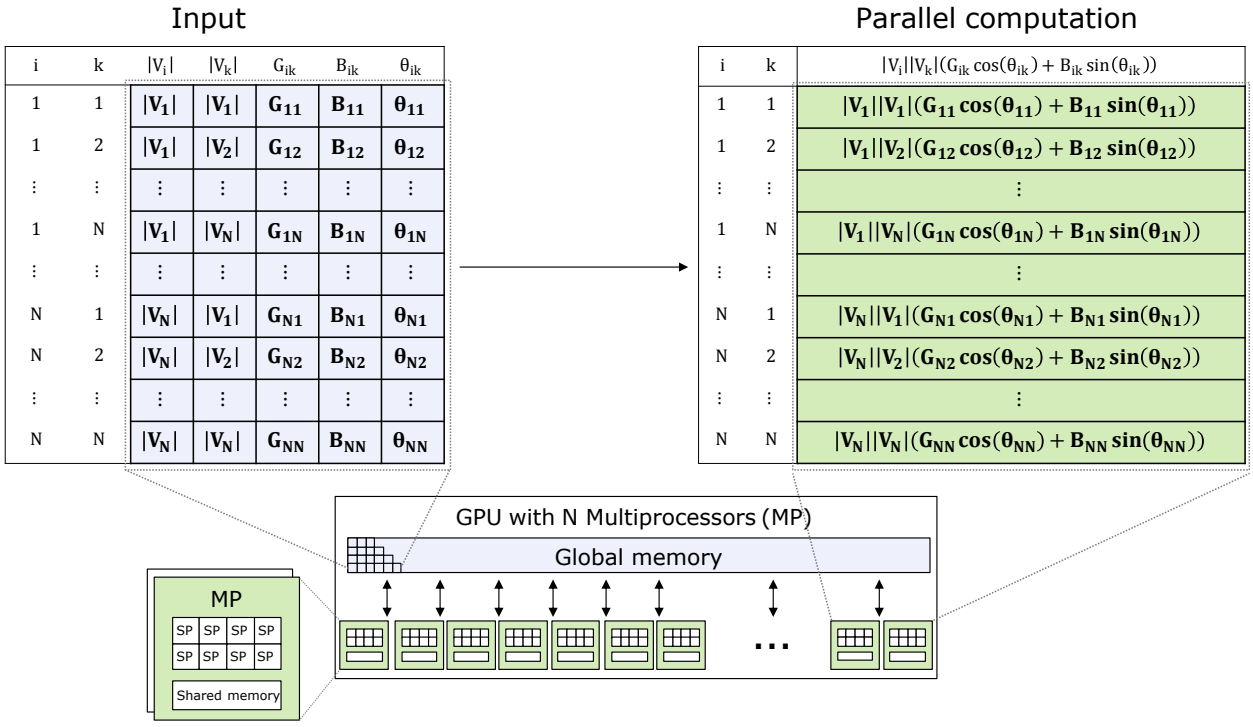

Figure 3-2: Definition of the required information as the input of the parallel computation

3.3 Power Flow Vectorization in Embedded Computer Using a GPU

The GPU architecture is used to parallelize specific power flow process that include repetitive calculations. Parallelism in the GPU is used to speed up $\mathbf{Y_{bus}}$ and $\mathbf{J}$ matrices and the $\mathbf{\Delta P}, \mathbf{\Delta Q}$, $\mathbf{P}$, and $\mathbf{Q}$ vectors. The methodology follows the general structure in [13].

Power flow strategies cannot be completely implemented in GPU since the not all structures are allowed to be effectively parallelizable. Algorithm 1 shows a classical Newton-Raphson sequentially executed. Green operations are vectorized to reduced large matrices formation and nodal balance calculations. The vectorized operations are firstly organized in data arrays during a pre-processing stage. Once data is pre-processed, GPU vectorized operations are computed in a GPU kernel. GPU results are requested from the CPU. The output data array are finally transferred from the GPU to CPU in the EC.

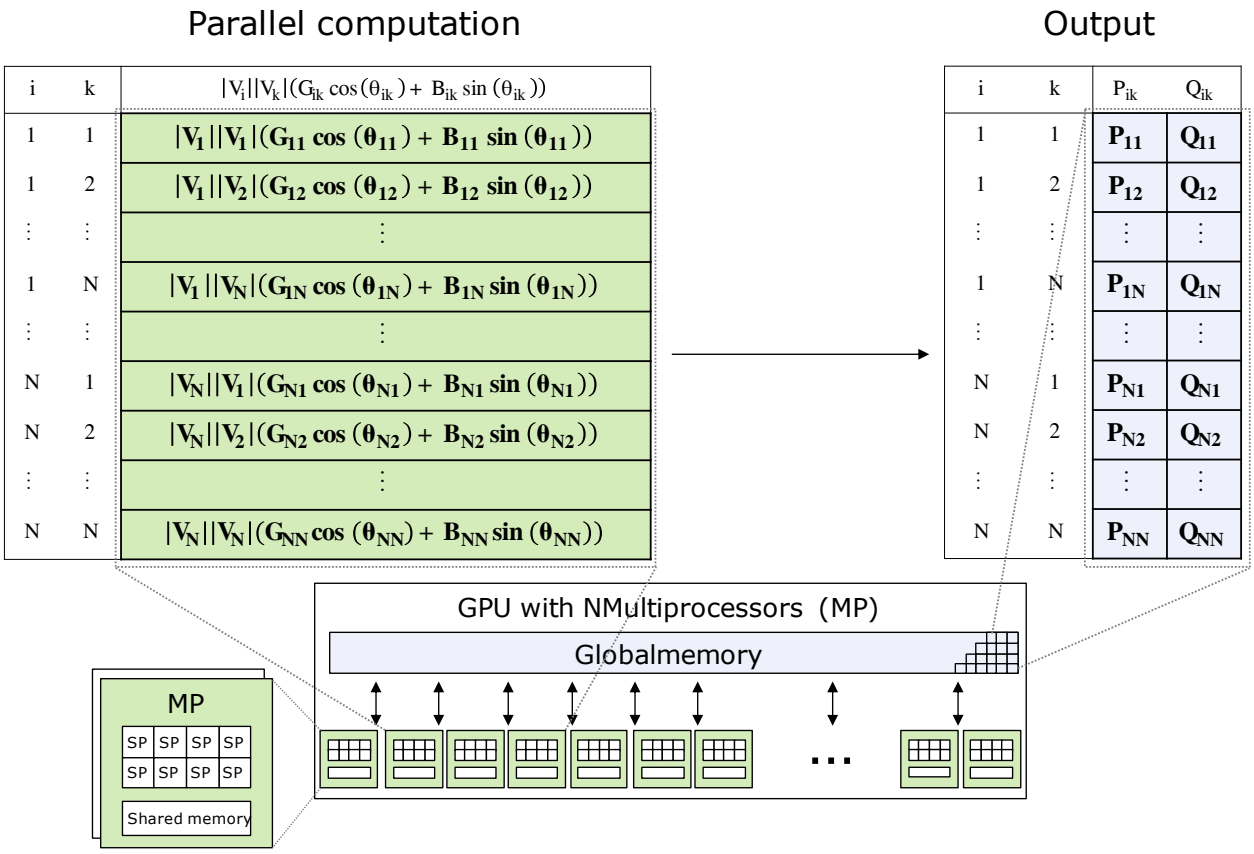

Figure 3-3: Parallel computation of defined instruction and output data

3.3.1 $\mathbf{Y_{bus}}$ Matrix Computation

Since admittance matrices in large power system have a large degree of sparsity, a Compressed Sparse Row Format (CSR) format to store sparse matrices is used. Data array for $\mathbf{Y_{bus}}$ matrix formation is shown in Table **3-1**. The array is used as input array in the GPU. R and X are the real and imaginary parts of each branch impedance, G_C and B_C are the conductance and susceptance of the buses, a is the transformer ratio, and l is the number of branches.

Table 3-1: Array input for computing $\mathbf{Y_{bus}}$ in GPU regarding the pre-processing stage

R_1	X_1	$Gc1$	$Bc1$	a_1
R_2	X_2	Gc_2	Bc_2	a_2
⋮	⋮	⋮	⋮	⋮
R_l	X_l	Gcl	Bcl	a_l

Table 3-2: Computation of the matrix $\mathbf{Y_{bus}}$ in GPU

$Ys = \frac{1}{R_1 + jX_1}$ $Y_{tt} = Ys + \frac{Gc_1 + jBc_1}{2}$ Y_{tt}/a_1^2 $-Ys/a_1$
$Ys = \frac{1}{R_2 + jX_2}$ $Y_{tt} = Ys + \frac{Gc_2 + jBc_2}{2}$ Y_{tt}/a_2^2 $-Ys/a_2$
$\vdots$
$Ys = \frac{1}{R_l + jX_l}$ $Y_{tt} = Ys + \frac{Gc_l + jBc_l}{2}$ Y_{tt}/a_l^2 $-Ys/a_l$

Once the data is transferred and organized in the GPU in data spaces that contain R, X, G_C, B_C and a as described in Table **3-1**. Calculations of the vectorized structure is performed using multiple GPU threads. Each thread calculates independent equations as shown in Table **3-2**. Y_s represents series impedance of each branch.

The output array is the full of calculations computed by each thread as shown in Table **3-3**. The impedance matrix $\mathbf{Y_{bus}}$ is formed by the assembled structure during the Newton Raphson algorithm execution. $Y_{tt}, Y_{ff}, Y_{ft}, Y_{tf}$ are the $\mathbf{Y_{bus}}$ elements for each branch.

Table 3-3: $\mathbf{Y_{bus}}$ GPU Kernel Output

Y_{tt_1}	Y_{ff_1}	Y_{ft_1}	Y_{tf_1}
Y_{tt_2}	Y_{ff_2}	Y_{ft_2}	Y_{tf_2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_{tt_l}	Y_{ff_l}	Y_{ft_l}	Y_{tf_l}

Table 3-4: **P**, **Q** and **J** kernel input

i_1	k_1	$G_{i_1k_1}$	$B_{i_1k_1}$	$\|E\|_{i_1}$	$\|E\|_{k_1}$	θ_{i_1}	θ_{jk_1}
i_2	k_2	$G_{i_2k_2}$	$B_{i_2k_2}$	$\|E\|_{i_2}$	$\|E\|_{k_2}$	θ_{i_2}	θ_{k_2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i_l	k_l	$G_{i_lk_l}$	$B_{i_lk_l}$	$\|E\|_{i_l}$	$\|E\|_{k_l}$	θ_{i_l}	θ_{k_l}

3.3.2 P, Q and J Computation

Algorithms that uses CPU-GPU structures have limitations in the data transfer process between CPU - GPU migration since bandwidth is a limiting factor in the hardware. This bottleneck generates considerable time delays in algorithm's execution. To reduce the data exchange process, the load flow analysis maximizes the GPU operations during each iteration. The algorithm performs one vectorized GPU kernel to calculate **P** and **Q** vectors, and the matrix **J** during each data transfer. Previous vectors and matrices are computed using the GPU input array presented in Table **3-4**. Once the input array is in the GPU memory, the kernel performs all different calculations in parallel using the existing GPU threads, solving the power flow equations as described in Table **3-5**. Each GPU thread performs independent calculations of each equation. GPU output is shown in Table **3-6**. Matrix **J** is assembled using the CSR format including the slack bus and other PV buses. Similarly, $\Delta\mathbf{P}$ and $\Delta\mathbf{Q}$ are calculated based on Equations 3-1 and 3-2. The results are presented in Table **3-6**. The power imbalances result of calculated and actual power in a bus are calculated in parallel.

3.4 Results and Discussion

The hybrid CPU-GPU algorithm is implemented in a low cost hardware infrastructure Nvidia Jetson Nano. This EC includes the features described in Table **3-7**.

NVIDIA CUDA framework and platform are the GPU structures with high maturity in terms of programming interfaces between graphic and non-graphic programming languages such as C, C++, and FORTRAN [112, 113]. Similarly, popular software as Python and Matlab included libraries and toolkits that make the interaction with NVIDIA GPUs a less complex task. The Algorithm 1 to manage multiple GPU cores is implemented using Numba module in Python programming language [114]. Sections in green from the LF-NR algorithm 1 empowered by the GPU kernels are parallellized using the structure in Figure **3-4**. GPU kernels allow the calculation of the vectors **P** and **Q** and the matrix **J**. The input *Data* refers to the array shown in Table **3-1**, while S corresponds to the output array

Table 3-5: **P**, **Q** and **J** kernel calculations

$\|E_{i_1}\|\|E_{k_1}\|\left[G_{i_1k_1}\sin\left(\theta_{i_1}-\theta_{k_1}\right)-B_{i_1k_1}\cos\left(\theta_{i_1}-\theta_{k_1}\right)\right]$
$\|E_{i_1}\|\left[G_{i_1k_1}\cos\left(\theta_{i_1}-\theta_{k_1}\right)+B_{i_1k_1}\sin\left(\theta_{i_1}-\theta_{k_1}\right)\right]$
$-\|E_{i_1}\|\|E_{k_1}\|\left[G_{i_1k_1}\cos\left(\theta_{i_1}-\theta_{k_1}\right)+B_{i_1k_1}\sin\left(\theta_{i_1}-\theta_{k_1}\right)\right]$
$\|E_{i_1}\|\left[G_{i_1k_1}\sin\left(\theta_{i_1}-\theta_{k_1}\right)-B_{i_1k_1}\cos\left(\theta_{i_1}-\theta_{k_1}\right)\right]$
$\|E_{i_1}\|\|E_{k_1}\|\left[G_{i_1k_1}\cos\left(\theta_{i_1}-\theta_{k_1}\right)+B_{i_1k_1}\sin\left(\theta_{i_1}-\theta_{k_1}\right)\right]$
$\|E_{i_1}\|\|E_{k_1}\|\left[G_{i_1k_1}\sin\left(\theta_{i_1}-\theta_{k_1}\right)-B_{i_1k_1}\cos\left(\theta_{i_1}-\theta_{k_1}\right)\right]$
$\|E_{i_2}\|\|E_{k_2}\|\left[G_{i_2k_2}\sin\left(\theta_{i_2}-\theta_{k_2}\right)-B_{i_2k_2}\cos\left(\theta_{i_2}-\theta_{k_2}\right)\right]$
$\|E_{i_2}\|\left[G_{i_2k_2}\cos\left(\theta_{i_2}-\theta_{k_2}\right)+B_{i_2k_2}\sin\left(\theta_{i_2}-\theta_{k_2}\right)\right]$
$-\|E_{i_2}\|\|E_{k_2}\|\left[G_{i_2k_2}\cos\left(\theta_{i_2}-\theta_{k_2}\right)+B_{i_2k_2}\sin\left(\theta_{i_2}-\theta_{k_2}\right)\right]$
$\|E_{i_2}\|\left[G_{i_2k_2}\sin\left(\theta_{i_2}-\theta_{k_2}\right)-B_{i_2k_2}\cos\left(\theta_{i_2}-\theta_{k_2}\right)\right]$
$\|E_{i_2}\|\|E_{k_2}\|\left[G_{i_2k_2}\cos\left(\theta_{i_2}-\theta_{k_2}\right)+B_{i_2k_2}\sin\left(\theta_{i_2}-\theta_{k_2}\right)\right]$
$\|E_{i_2}\|\|E_{k_2}\|\left[G_{i_2k_2}\sin\left(\theta_{i_2}-\theta_{k_2}\right)-B_{i_2k_2}\cos\left(\theta_{i_2}-\theta_{k_2}\right)\right]$
$\vdots$
$\|E_{i_2}\|\|E_{k_2}\|\left[G_{i_lk_l}\sin\left(\theta_{i_l}-\theta_{k_l}\right)-B_{i_lk_l}\cos\left(\theta_{i_l}-\theta_{k_l}\right)\right]$
$\|E_{i_l}\|\left[G_{i_lk_l}\cos\left(\theta_{i_l}-\theta_{k_l}\right)+B_{i_lk_l}\sin\left(\theta_{i_l}-\theta_{k_l}\right)\right]$
$-\|E_{i_l}\|\|E_{k_l}\|\left[G_{i_lk_l}\cos\left(\theta_{i_l}-\theta_{k_l}\right)+B_{i_lk_l}\sin\left(\theta_{i_l}-\theta_{k_l}\right)\right]$
$\|E_{i_l}\|\left[G_{i_lk_l}\sin\left(\theta_{i_l}-\theta_{k_l}\right)-B_{i_2k_l}\cos\left(\theta_{i_l}-\theta_{k_l}\right)\right]$
$\|E_{i_l}\|\|E_{k_l}\|\left[G_{i_lk_l}\cos\left(\theta_{i_l}-\theta_{k_l}\right)+B_{i_lk_l}\sin\left(\theta_{i_l}-\theta_{k_l}\right)\right]$
$\|E_{i_l}\|\|E_{k_l}\|\left[G_{i_lk_l}\sin\left(\theta_{i_l}-\theta_{k_l}\right)-B_{i_lk_l}\cos\left(\theta_{i_l}-\theta_{k_l}\right)\right]$

Table 3-6: **P**, **Q** and **J** kernel output

$\frac{\partial P_i}{\partial\theta_k}$	$\frac{\partial P_i}{\partial\|E_k\|}$	$\frac{\partial Q_i}{\partial\theta_k}$	$\frac{\partial Q_i}{\partial\|E_k\|}$	P_{i_1}	Q_{i_1}
$\frac{\partial P_i}{\partial\theta_k}$	$\frac{\partial P_i}{\partial\|E_k\|}$	$\frac{\partial Q_i}{\partial\theta_k}$	$\frac{\partial Q_i}{\partial\|E_k\|}$	P_{i_2}	Q_{i_2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{\partial P_i}{\partial\theta_k}$	$\frac{\partial P_i}{\partial\|E_k\|}$	$\frac{\partial Q_i}{\partial\theta_k}$	$\frac{\partial Q_i}{\partial\|E_k\|}$	P_{i_l}	Q_{i_l}

Table 3-7: Jetson Nano Technical Features

GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
CPU	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	16 GB eMMC 5.1
Camera	12 lanes (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 (1.5 Gb/s per pair)
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI 2.0 and eDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I2C, I2S, SPI, UART
Mechanical	69.6 mm x 45 mm 260-pin edge connector

presented in Table **3-3**. SciPy library [115] allows the computation of the inverse of the Jacobian matrix included in the algorithm in Figure **3-4**.

The hybrid CPU-GPU algorithm response time is aligned with on-line power system applications. The LF-NR algorithm is benchmark using the MATPOWER package [116] comparing computation time and power flow results. Test systems included IEEE30, Case300, 2869 Pegase, 9241 Pegase and ACIVSg25k included in MATPOWER library. The MATPOWER performance is evaluated in a PC with an AMD Rayzen™ 5 3600 with six cores and 12 logical processors. To ensure that the algorithm performance using the NVIDIA Jetson Nano fits properly with online power system applications, the LF-NR method provided by the MATLAB-language power system simulation package MATPOWER [116] is employed to compare the computation time and the result of the algorithm. The test systems correspond to the cases IEEE30, IEEE300, 2869PEGASE, 9241PEGASE, and ACIVSg25k provided by MATPOWER. The MATPOWER performance is evaluated in a PC with an AMD Rayzen™ 5 3600 with six cores and 12 logical processors. The LF-NR in Algorithm 1 implemented in Python is computed in MATPOWER using only CPU memory without interaction with GPU. Figure **3-5** shows the power balance error ϵ for the test cases. The error is negligible (less than 1×10^{-13} difference in the results) for MATPOWER and Python running in the PEC and EC.

Five different test cases systems were used to compare the LF-NR algorithm performance using the NVIDIA Jetson Nano, the AMD Rayzen™ 5 3600 CPU with Python and MATPOWER. Figure **3-6** shows the computation time for the three platforms. The results shows that large power systems e.g. 25,0000 buses (ACIVSg25k) are solved in 3.1 s. On the other hand, small systems e.g. 30 buses (IEEE30) are solved in 0.1 s.

```
@guvectorize( [(float64[:],float64[:])],   '(n)->(n)',target='cuda')
def P_Q_J_Kernel(Data,S):
    E_i      = Data[0]
    E_k      = Data[1]
    D_theta  = Data[2]-Data[3]
    G_ik     = Data[4]
    B_ik     = Data[5]
    sin_d    = sin(D_theta)
    cos_d    = cos(D_theta)
    # Jacobian elements
    S[0]   = E_i*E_k*(G_ik*sin_d-B_ik*cos_d)
    S[1]   = (G_ik*cos_d+B_ik*sin_d)*E_i
    S[2]   = -S[1]*E_k
    S[3]   = S[0]/E_k
    # Pi and Qi
    S[4]   = E_i*E_k*(G_ik*cos_d+B_ik*sin_d)
    S[5]   = E_i*E_k*(G_ik*sin_d-B_ik*cos_d)
```

Figure 3-4: Cuda Kernel for computing $\mathbf{P}, \mathbf{Q}, \mathbf{J}$ matrix using Numba

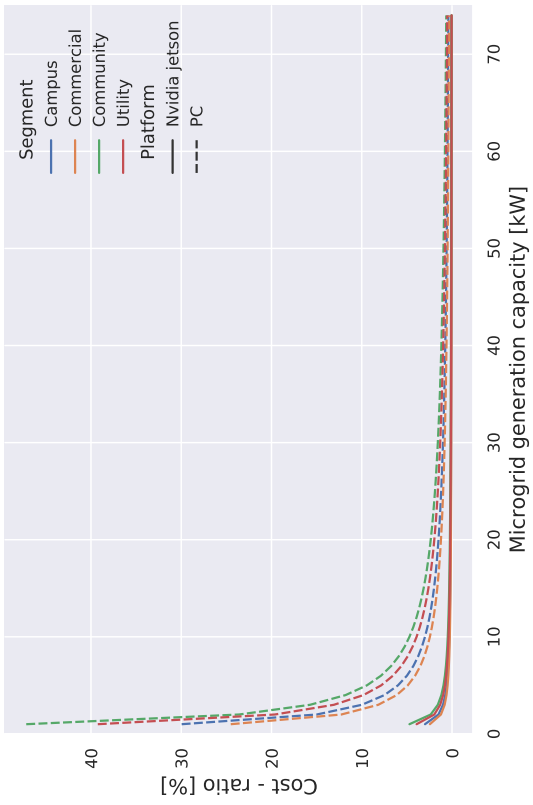

Figure 3-7: Cost-ratio comparison between the platform's cost and the microgrid's total cost regarding the microgrid generation capacity for different segments

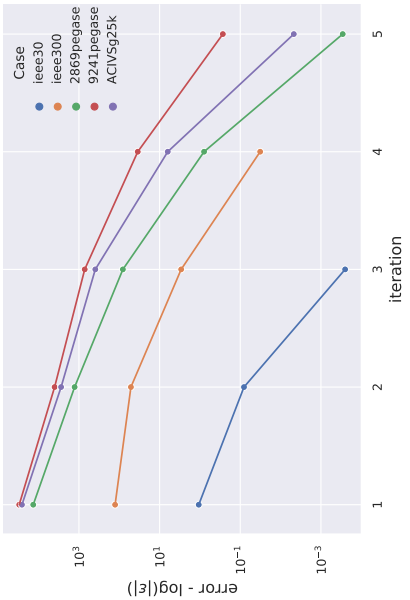

Figure 3-5: Nodal power balance using the developed algorithm run on Nvidia-Jetson Nano

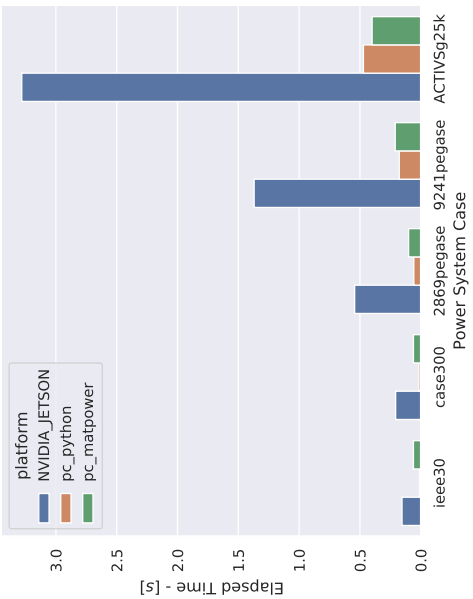

Figure 3-6: Time elapsed for load flow convergence using the developed algorithm run on Nvidia-Jetson Nano

Figure **3-6** shows that NVIDIA Jetson Nano computation time is higher than other platforms. These kind of responses are expected considering the EC and PEC technical features. EC calcultation time is on average 10 times slower than PEC. Nevertheless, the price difference between the PC and NVIDIA Jetson platforms is approximately 10 times whitouth including the licensing cost of MATLAB required for MATPOWER execution. Similarly, energy efficiency is higher NVIDIA Jetson Nano when the LF-NR algorithm is used for on-line calculations. All computation times in the five test cases fit properly for on-line operation and control of power grids.

Figure **3-6** also shows that the proposed vectorized LF-NR algorithm in Pyhton results in as faster response than MATPOWER. The proposed vetorization using a Numba library in the AMD Rayzen™ 5 3600 CPU is computationally more efficient than the LF-NR function in MATPOWER.

Authors in [105] reported the ratio for different generation sizes in diverse microgrids as seen in Figure **3-7**. Figure **3-7** shows the ratio between platform's cost and the microgrid's total cost for different microgrids capacities. Two platforms are used for comparison the AMD Rayzen™ 5 3600 CPU and the NVIDIA Jetson Nano. An average cost of $1,000 and $100 USD was used for EC and PEC. For small generation capacity PEC plays a relevant role in the total microgrid cost. Nevertheless, as the system capacity increases, the PEC cost impact is reduced. E.g. For a 5 kW of generation capacity, the cost-ratio is between 4.9 % and 9.4 % depending on the total size. For a similar generation capacity, the cost-ratio is between 0.5 % and 1 % of the microgrid's total cost for a EC platform. For 60 kW, the cost-ratio is between 0.4 % and 0.8 % for a PEC and 0.05 % and 0.08 % for an EC.

3.5 Conclusions

The low-cost strategy described in the chapter is suitable for online load flow analysis in power grids, specially microgrids. The Newton-Raphson is partially implemented vectorized in an Embedded Computer (EC). The EC in the implementation was a NVIDIA Jetson Nano empowered with a Graphical Processing Unit (GPU) to process the vectorized sections. The computation times in the EC are within 3 s for large power systems (e.g. 25,000 buses) making the strategy a cost-effective solution appropriate for an online analysis.

4 Security Constraint Optimal Power Flow Formulation and Solutions with Constraint Handling

The Security Constrained Optimal Power Flow (SCOPF) problem focuses on solving a power system operation optimization problem, while maintaining network security. In this way, the network not only operates at the lowest cost or highest benefit, but also guarantees that it continues its normal operation when one or more outages appear are presented. Grid codes normally request to satisfy (N-k criteria) with all variables within acceptable limits as described in Chapter 2 [117–121]. Multiple approaches have been proposed to solve this highly complex problem. The difficulty of the problem is a result of its nature, which describes it as a non-linear, non-convex, static problem, optimization of a large system, which normally includes integer variables such as taps position in transformers, switching of capacitor and reactor banks, switching of demand blocks when demand response is allowed in the grid [119, 120, 122, 123]. This type of problem is described as Mixed Integer Linear Programming (MILP).

Different authors have proposed various strategies to make the SCOPF problem have a more simplified implementation for computer simulation. Some approaches include modifications to the formulation of the optimization problem in terms of the objective function and constraints, among these are: linearization, the simplification to guarantee problme's convexity[1], the decomposition into more simplified and computationally individual problems (eg Alternating Direction Multipliers Method (ADMM), Augmented Lagragian Method (ALM), Benders Decomposition (BD), etc.). On the other hand, other approaches include reducing the feasible space, such is the case of alternatives such as contingency screening when systems have a large number of elements that can go out of operation. These strategies allow the simplification of the problem, yielding results with a normally acceptable level of precision and reduced simulation times. However, the alternatives have been tested in networks with few buses, lines, loads and generators. Some applications have been adapted to large networks for real-time execution; nevertheless, some deviations, oversimplifications of the original problem have limited its

[1]Simplification from Complex Problems to Quadratic Problem Formulation

implementation.

This chapter shows an alternative to solve a SCOPF problem for medium and large networks. This chapter describes an implemented fast decomposition strategy for the SCOPF problem. The algorithm allows modifications to the feasible problem's space adding or modifying the set of existing constraints. The algorithm is executed in parallel, taking advantage of the computational potential in computer cores at the local level, or computer cluster in group of multiple nodes. The benefits of implementation include:

- Modifications of the solution space, adding or modifying constraint of the SCOPF problem.
- Adding Spin Reserve constraints as well as AGC control.
- Filtering contingencies depending on the system operation state.
- Decomposition and addition of constraints based on parallel PFs that increase the simulation speed and reduce problem's complexity of simultaneous OPF problems for hundreds of contingencies.
- Generation re-dispatch showing a PSCOPF problem.
- PV / PQ switching to ensure convergence in large power networks.

The chapter presents: some strategies to solve the problem, the complete formulation of the SCOPF problem, explains the approach used to solve it, shows the results obtained, conducts discussions about the algorithm and finally concludes.

4.1 Security-constrained OPF: An Overview

In order to reduce the operating costs of interconnected power systems, regulatory entities such as North American Electric Reliability Corporation (NERC) and Federal Energy Regulatory Commission (FERC) have highlighted the importance of operating the grid in an optimal and secure manner. These entities have mentioned that power system operation should not be optimized only in the long term, but also in the short term or near real time [122, 124, 125]. In [125], it is mentioned that the annual savings associated with an improvement of 5% of the OPF algorithms, could lead to improvements of up to six billion dollars annually in United States.

Nevertheless, such improvements are not easily achievable due to the complexity offered by the SCOPF formulation. This complexity becomes greater when the system size increases, resulting in high computational burden to solve systems with a large number of elements and contingencies [126, 127]. This makes Nonlinear Programming Problems-Hard Nonlinear Programming Problems-Hard (NP-Hard) one of the most difficult cases [128]. This condition reduces the possibility of finding global optimum using the complete formulation of the SCOPF problem in limited time [118]. Some strategies to reduce problem complexity and get a solution in short time are shown in the rest of the section.

4.1.1 Categories of SCOPF Problem

As described in Chapter 2, there are two variants of SCOPF that allow the constraints associated with contingencies to be included. The PSCOPF [121, 123] makes adjustments to the pre-contingency case in such a way that it is possible to carry out redispatches or adjustments to control variables (eg. power of the generators that participate in the infrequency control, automatic tap-changers, switching of power banks, reactors and secondary voltage control) [119, 127], in order to guarantee that the secure operating limits are not exceeded in the different electrical elements. Some examples of PSCOPF are shown in [126, 129], showing over-tightened solution regions and long computation times due to the high number of contingencies [121, 123, 127, 130, 131]. On the other hand, the CSCOPF allows adjustments to the generation dispatch and other control variables. The control is not only applied in the pre-contingency state but also after the operation of one of the elements of the power system. Examples of CSCOPF implementation can be found in [118, 123, 126]. However, most of these works have shown limited proposals that include active power droop control, the PV / PQ switching, the lack of modeling of power system controllers and neither the cost of remedial actions.

Other models include additional variables, such as risk assessment [117, 132, 133], constraints associated with the solution time [134], and stochastic modelling [130, 135]. The above requirements include computational time limitations and memory management when used in medium-sized and large-scale power systems, as well as with considerable numbers of contingencies [123, 135].

4.1.2 SCOPF Solution Strategies

4.1.2.1 Linearization and Convexification

Linearization techniques are commonly used to simplify nonlinear problems. In the case of power systems, DC simplifications are normally applied. This technique has been

commonly applied to solve problems [124, 129, 132, 136]. Although the approach works well in small systems, with low reactive power flows, a significant absence of voltage control, a lower number of reactive shunt elements or conditions with high load flows [119], they are inefficient in systems that do not present these conditions. Most of the approximations offer modifications on the solution methods such as Successive Linear Programming (SLP) [118], others act on the objective function of the problem. Other proposals include modifications to the existing information, creating quasi-real measurements that allow decoupling the solutions in active and reactive power. The previous strategy allows to accelerate the solution of the quadratic optimization problems [137].

4.1.2.2 Decomposition Strategies

Decomposition strategies focus on dividing a problem into small simplified subproblems with simpler solutions. Among the most popular strategies are: the Augmented Lagragian Method (ALM), the Alternating Direction Multipliers Method (ADMM) the Benders Decomposition (BD).

In OPF, ALM has been used to solve distributed systems [138]. In [139], the authors propose ALM to solve a reactive flow OPF for a large electrical network that has been distributed. However, applications of ALM with SCOPF is not reported.

In [132, 138, 140], ADMM is used to divide a complex problem and parallelize it in such a way that the simultaneous solution of an OPF problem is allowed. In [121] authors propose problem solution in a system with 3,012 nodes and 4 contingencies. The problem is solved in parallel simultaneously in 3,582 seconds. This computation time is not promising for larger power systems with a greater number of contingencies.

BD has been applied in various power system optimization problems. In [126] a strategy to solve a CSCOPF is applied; however, the algorithm is apply to very small power networks e.g. only 6 buses. In [117, 122] larger systems of 118 and 2,351 nodes were solved. However, DC simplifications of the power flow equations were used. In [121] a BD model is used to solve networks of more than 3,000 nodes; however, the algorithm took 1,165 seconds with only 4 contingencies. [141] shows results for larger power networks of 2,312 and 3,013 nodes and 990 contingencies using the BD method.

4.1.2.3 Optimization Techniques

Both the OPF and SCOPF problems are non-convex, eliminating the idea of finding a global optimum mathematically [142]. Various approaches have been carried out in order

to explore extensive solution spaces, including Genetic Algorithm (GA), Metaheuristic Algorithm (MA) and Machine Learning (ML). Some strategies using optimization algorithms for OPF solution include: earthworms, fuzzy logic tuned firefly, and historical data-based approaches are presented in [143–145]. Although the algorithms find global optimum, they are used in small power system, reaching 300 buses. Similarly, the authors mention that these algorithms do not ensure a defined number of iterations to find a solution.

Different authors have made hybrid proposals that integrate not only exact mathematical formulations but also searches with MA. In [142] an OPF problem is solved by means of GA that group chromosomes, to later use a precise method such as Newton-Raphson in order to have a good initial solution point and later apply a precise method. The algorithm reduces existing constraints by not considering overhead limits. In [123] a simplified problem is solved through a DC approximation using ALM and ADMM algorithms. BD is used to solve a system using an Evolutionary Algorithm (EA) for contingency selection in [130]. However, its use is reported in small systems of 118 nodes.

4.1.2.4 Contingency Screening

Some alternatives to accelerate the SCOPF problem's solutions as well as reducing the problem size, is using a contingency filtering. In this type of strategy, the number of possible contingencies that may appear in a system is reduced, highlighting the most severe ones or those that restrict the secure power system response as shown in [119, 120]. In [146] the umbrella contingency method is used, which generates a contingency ranking according to contingency impact on the grid. The impact is measured by the magnitude of Lagrangian multipliers associated with the post-contingency states. The method requires the solution of the complete SCOPF problem to generate the contingency ranking. Solving the complete SCOPF solution makes the proposal unfeasible for real-time application.

Other authors propose diverse algorithms in real-time. In [147] a weighted method and a central eigenvector of the Laplacian matrix method is proposed. The method adds elements to the Laplacian matrix based on line overloads that may result after an outage. When high number of contingencies appear, a considerable number of elements of the matrix are fill out, showing long computation times for real time. On the other hand, [130] uses EA to perform contingency filtering; however, the algorithm demands at least one complete iteration to identify contingencies that affect network security.

In [118], a contingency filtering was performed according to the criticality and impact index for each single contingency. The contingency rank is defined in terms of system

impact. Nevertheless, the method requires multiple power flow solutions for all the set of contingencies, making its application unfeasible in real time.

4.2 Formulation of the SCOPF Problem

4.2.1 Complete Formulation

In this chapter the SCOPF is focus on cost C_{tot} minimization as shown in:

$$min(C_{tot}) = min\left[\left(\sum_{g\in G} c_g\right) + \delta c^{\sigma} + \frac{1-\delta}{|K|}\sum_{\substack{k\in SC\\ k\neq 0}} c_k^{\sigma}\right] \tag{4-1}$$

Where:

G = Set of generators
c_g = Generation cost of generator g
c^{σ} = Total constraint violation penalty in base case
c_k^{σ} = Total constraint violation penalty in contingency k
K = Set of all contingencies
δ = Weight assigned to the penalty cost in the base case

Lower and upper bounds for different variables x are defined as $\underline{x}$ and $\overline{x}$

$$SC = \{0, 1, 2, 3, ..., |K| - 1, |K|\} \tag{4-2}$$

Where:

sc = Particular scenario of the set SC
sc_0 = Base case
sc_i = i-th contingency scenario

In this chapter, contingency are associated with outage of a line, transformer, capacitor bank or generator at a time ($N-1$ contingency).

Bound variables for voltage, active and reactive power are shown below:

$$\underline{v_i} \leq v_i \leq \overline{v_i} \quad \forall i \ \in \ I^{sc} \ \wedge \forall sc \ \in \ SC \tag{4-3}$$

Where:

I^{sc} = Set of active buses in scenario sc.
v_i = Voltage magnitude on bus i.

$$\underline{p_g} \leq p_g \leq \overline{p_g} \quad \forall g \in G^{sc} \wedge \forall sc \in SC \tag{4-4}$$

$$\underline{q_g} \leq q_g \leq \overline{q_g} \quad \forall g \in G^{sc} \wedge \forall sc \in SC \tag{4-5}$$

Where:

G^{sc} = Set of active generators in scenario sc
p_g = Active Power of generator g
q_g = Reactive power of generator g

Line overloads constraints are shown below:

$$\sqrt{(p_e^o)^2 + (q_e^o)^2} \leq \overline{R}_e v_{i_e}^o + \sigma_e^{sc,s} \quad \forall e \in E^{sc} \wedge \forall sc \in SC \tag{4-6}$$

$$\sqrt{(p_e^d)^2 + (q_e^d)^2} \leq \overline{R}_e v_{i_e}^d + \sigma_e^{sc,s} \quad \forall e \in E^{sc} \wedge \forall sc \in SC \tag{4-7}$$

$$\sigma_e^{sc,s} \geq 0 \; \forall sc \in SC \tag{4-8}$$

Where:

p_e^o = Active power from origin bus of the line e
q_e^o = Reactive power from origin bus of the line e
p_e^d = Active power to destination bus of the line e
q_e^d = Reactive power to destination bus of the line e
E^{sc} = Set of active transmission lines in scenario sc
$v_{i_e}^o$ = Voltage magnitudes at the origin bus in the line e
$v_{i_e}^d$ = Voltage magnitudes at the destination buses respectively for the line e
$\overline{R}_e$ = Maximal allowed current in line e expressed in MVA at rated voltage
$\sigma_e^{sc,s}$ = Slack variable that accommodates the current in excess of line e's capacity

$\sigma_e^{sc,s}$ is used for penalty calculation, for scenario sc

Transformer constraint are shown below:

$$\sqrt{(p_f^o)^2 + (q_f^o)^2} \leq \overline{s}_f + \sigma_f^{sc,s} \quad \forall f \in F^{sc} \wedge \forall sc \in SC \tag{4-9}$$

$$\sqrt{(p_f^d)^2 + (q_f^d)^2} \leq \overline{s}_f + \sigma_f^{sc,s} \quad \forall f \in F^{sc} \wedge \forall sc \in SC \tag{4-10}$$

$$\sigma_e^{sc,s} \geq 0 \ \forall sc \ \in \ SC \tag{4-11}$$

$$\sigma_f^{sc,s} \geq 0 \ \forall sc \ \in \ SC \tag{4-12}$$

Where:

p_f^o = Active power from origin bus of the transformer f
q_f^o = Reactive power from origin bus of the transformer f
p_f^d = Active power to destination bus of the transformer f
q_f^d = Reactive power to destination bus of the transformer f
F^{sc} = Set of active transformer branches in scenario sc
$v_{i_f}^o$ = Voltage magnitudes at the origin bus in the transformer f
$v_{i_f}^d$ = Voltage magnitudes at the destination buses respectively for the transformer f
$\overline{S}_f$ = Maximal allowed current in line f expressed in MVA at rated voltage
$\sigma_f^{sc,s}$ = Slack variable that accommodates the current in excess of transformer f's capacity

$\sigma_f^{sc,s}$ is used for penalty calculation, for scenario sc.

To avoid discrete models for reactive devices, shunt capacitor or reactors are considered as generators with $P_g = 0$.

$$\underline{b_i^{cs}} v_i^2 \leq q_i^{cs} \leq \overline{b_i^{cs}} v_i^2 \quad \forall i \ \in \ I^{sc} \ \wedge \forall sc \ \in \ SC \tag{4-13}$$

Where:

q_i^{cs} = Reactive power of commutable shunt on bus i
b_i^{cs} = Susceptance value of the commutable shunt on bus i
v_i = Voltage magnitude on the bus i.

Active and reactive power balance constraints are shown below:

$$\begin{aligned} &\sum_{g \in G_i^{sc}} p_g - p_{L_i}^{sc} - g_{fs_i}^{sc} v_i^2 - \sum_{e \in E_i^{sc,o}} p_e^o - \\ &\sum_{e \in E_i^{sc,d}} p_e^d - \sum_{f \in F_i^{sc,o}} p_f^o - \sum_{f \in F_i^{sc,d}} p_f^d = \\ &\sigma_i^{sc,P+} - \sigma_i^{sc,P-} \ \forall i \ \in \ I^{sc} \ \wedge \forall sc \ \in \ SC \end{aligned} \tag{4-14}$$

$$\sigma_i^{sc,P+} \geq 0 \ \forall i \ \in \ I^{sc} \ \wedge \forall sc \ \in \ SC \tag{4-15}$$

$$\sigma_i^{sc,P-} \geq 0 \; \forall i \; \in \; I^{sc} \; \wedge \forall sc \; \in \; SC \tag{4-16}$$

$$\begin{aligned} &\sum_{g \in G_i^{sc}} q_g - q_{L_i}^{sc} - (-b_{fs_i}^{sc} - b_{cs_i}^{sc}) v_i^2 - \sum_{e \in E_i^{sc,o}} q_e^o - \\ &\sum_{e \in E_i^{sc,d}} q_e^d - \sum_{f \in F_i^{sc,o}} q_f^o - \sum_{f \in F_i^{sc,d}} q_f^d = \\ &\sigma_i^{sc,Q+} - \sigma_i^{sc,Q-} \; \forall i \; \in \; I^{sc} \; \wedge \forall sc \; \in \; SC \end{aligned} \tag{4-17}$$

$$\sigma_i^{sc,Q+} \geq 0 \; \forall i \; \in \; I^{sc} \; \wedge \forall sc \; \in \; SC \tag{4-18}$$

$$\sigma_i^{sc,Q-} \geq 0 \; \forall i \; \in \; I^{sc} \; \wedge \forall sc \; \in \; SC \tag{4-19}$$

Where:

$p_{L_i}^{sc}$ = Active load power on bus i in scenario sc
$q_{L_i}^{sc}$ = Reactive load power on bus i in scenario sc
$g_{fs_i}^{sc}$ = Conductance of the fixed shunts on bus i in scenario sc
$b_{fs_i}^{sc}$ = Susceptance of the fixed shunts on bus i in scenario sc
$E^{sc,o}$ = Set of active lines in scenario sc that have bus i as the origin bus
$E^{sc,d}$ = Set of active lines in scenario sc that have bus i as the destination bus
$F^{sc,o}$ = Set of active transformers in scenario sc that have bus i as the origin bus
$F^{sc,d}$ = Set of active transformers in scenario sc that have bus i as the destination bus
$\sigma_i^{sc,P+}$ = Slack variables for the positive parts of the violation of P balance for bus i in sc.
$\sigma_i^{sc,P-}$ = Slack variables for the negative parts of the violation of P balance for bus i in sc.
$\sigma_i^{sc,Q+}$ = Slack variables for the positive parts of the violation of Q balance for bus i in sc.
$\sigma_i^{sc,Q-}$ = Slack variables for the negative parts of the violation of Q balance for bus i in sc.

Lack of power is avoided using a spin reserve constraint that guarantees load supplied during a generation outage.

$$\sum_{g \in A} \left(\overline{P}_g - P_g \right) + \sigma_A \geq max_{g \in (\chi \bigcap A)} \overline{P}_g \tag{4-20}$$

Where:

χ = Set of all generators that appear in at least one contingency
σ_A = An area spin reserve slack variable for each affected area A in the base case

Penalization cost in Eq. 4-1 is computed as shown below:

$$\begin{aligned} C_{sc}^{\sigma} = & \sum_{i \in I} \alpha_p \left(\sigma_i^{sc,P+} + \sigma_i^{sc,P-} \right) + \sum_{i \in I} \alpha_q \left(\sigma_i^{sc,Q+} + \sigma_i^{sc,Q-} \right) \\ & + \sum_{f \in F} \alpha_{ef} \left(\sigma_f^{sc,s} \right) + \sum_{e \in E} \alpha_{ef} \left(\sigma_e^{sc,s} \right) + \alpha_{\sigma} \sigma_{A,sc} \end{aligned} \tag{4-21}$$

Constraint violations that arise in the pre- and post-contingency cases are penalized following a piece-wise linear function. The violations are divided into: small, medium and large depending on the violation size. This penalty cost was related to soft constraints, to force feasible problem's solution using slack variables. The soft constraints are related with lines and transformers overloads, that appear when current and apparent power limits are exceeded. On the other hand, voltages in buses and the active and reactive power in generators were associated with hard constraints. Violations of the hard constraints, makes the problem infeasible.

On the other hand, in order to keep the frequency within the expected limits, a power balance constraint is included. Secondary regulation allows generation balance inside each zone using the generation droop of certain machines. The droop modeling is included through predefined participation factors defined by the network operator [148]. Once a generation contingency has occurred, the system must guarantee that the existing power generated in the area is sufficient to satisfy the demand in the area.

$$P_g^{sc} = \begin{cases} \underline{P_g} & \text{if } P_g^0 + a_g \Delta P^{sc} \leq \underline{P_g} \\ P_g^0 + a_g \Delta P^{sc} & \text{if } \underline{P_g} \leq P_g^0 + a_g \Delta P^{sc} \leq \overline{P_g} \\ \overline{P_g} & \text{if } P_g^0 + a_g \Delta P^{sc} \geq \overline{P_g} \end{cases} \tag{4-22}$$

Where:

P_g^{sc} = Real power of the g generator in the affected area to the sc contingency
a_g = Participation factor of the generator g
P_g^0 = Active power of the generator g in the base case
ΔP^{sc} = Difference of real power generation between the base case and the sc contingency

The handling of PV/PQ switching is a quite common problem in large scale power grids. The strategy implemented includes the use of voltage stability limit criteria when a $sc > 0$. The strategy keeps the voltage magnitude to the original voltage level before the outage is applied. However, if reactive limits violation appears, a transition to PQ is performed [149]:

$$\begin{cases} V_g^{sc} = V_g^0 & \text{if } \underline{Q_g} \leq Q_g^{sc} \leq \overline{Q_g} \\ \underline{V_g} \leq V_g^{sc} \leq V_g^0 & \text{if } \overline{Q_g^{sc}} = \overline{Q_g} \\ \overline{V_g^0} \leq V_g^{sc} \leq \overline{V_g} & \text{if } Q_g^{sc} = \underline{Q_g} \end{cases} \tag{4-23}$$

Where:

Q_g^{sc} = Reactive power generated variable in the contingency sc
V_g^0 = Voltage magnitude variables in the generators in the base case
V_g^{sc} = Voltage magnitude variables in the generators in contingency sc

4.2.2 Proposed Approach

Simultaneously considering all the existing contingencies in a system is a high computational demanding task. This problem increases in complexity when large power systems are evaluated. With large power systems, not only the matrix size increases but also the dispersed structure. This fact substantially increases the linear equations solution times.

To simplify the SCOPF problem that could result when including the base case and constraints and the N contingency states, and their constraints, the propose strategy includes constraint handling in the base case. In this way, post-contingency cases are evaluated and existing violations are observed in the new scenarios. Once the constraints are generated, the existing constraints of the base case are modified iteratively without generating new constraints or expanding the size and complexity of the base case problem. According to this strategy, the objective function of the base case is:

$$C = \sum_{g \in G} C_g + \delta C^{\sigma} \tag{4-24}$$

$\underline{x}(m)$ and $\overline{x}(m)$ are used to denote lower and upper bounds of the variable x during each m iteration. The iterative process for constraints update appears below:

$$\underline{v_i}(m) \leq v_i \leq \overline{v_i}(m) \quad \forall i \ \in \ I \tag{4-25}$$

$$\underline{p_g}(m) \leq p_g \leq \overline{p_g}(m) \quad \forall g \ \in \ G \tag{4-26}$$

$$\underline{q_g}(m) \leq q_g \leq \overline{q_g}(m) \quad \forall g \ \in \ G \tag{4-27}$$

$$\sqrt{(p_e^o)^2 + (q_e^o)^2} \leq \overline{R}_e(m) v_{i_e}^o + \sigma_e^s \quad \forall e \ \in \ E \tag{4-28}$$

$$\sqrt{(p_e^d)^2 + (q_e^d)^2} \leq \overline{R}_e(m) v_{i_e}^d + \sigma_e^s \quad \forall e \ \in \ E \tag{4-29}$$

$$\sqrt{(p_f^o)^2 + (q_f^o)^2} \leq \overline{s}_f(m) + \sigma_f^s \quad \forall f \ \in \ F \tag{4-30}$$

$$\sqrt{(p_f^d)^2 + (q_f^d)^2} \leq \overline{s}_f(m) + \sigma_f^s \quad \forall f \in F \tag{4-31}$$

$$\underline{b_i^{cs}} v_i^2 \leq q_i^{cs} \leq \overline{b_i}^{cs} v_i^2 \quad \forall i \in I \tag{4-32}$$

$$\begin{aligned} &\sum_{g \in G_i} p_g - p_i^L - g_i^{FS} v_i^2 - \sum_{e \in E_i^0} p_e^o - \sum_{e \in E_i^d} p_e^d - \\ &\sum_{f \in F_i^o} p_f^o - \sum_{f \in F_i^d} p_f^d = \sigma_i^{P+} - \sigma_i^{P-} \end{aligned} \tag{4-33}$$

$$\begin{aligned} &\sum_{g \in G_i} q_g - q_i^L - (-b_i^{FS} - b_i^{CS}) v_i^2 - \sum_{e \in E_i^0} q_e^o - \sum_{e \in E_i^d} q_e^d - \\ &\sum_{f \in F_i^o} q_f^o - \sum_{f \in F_i^d} q_f^d = \sigma_i^{Q+} - \sigma_i^{Q-} \end{aligned} \tag{4-34}$$

$$\sum_{g \in A} (\overline{P}_g(m) - P_g) + \sigma_A \geq max_{g \in (\chi \bigcap A)} \overline{P}_g \tag{4-35}$$

Slack variables magnitudes is used for penalization cost computation:

$$\begin{aligned} C^\sigma = &\sum_{i \in I} \alpha_p(\sigma_i^{P+} + \sigma_i^{P-}) + \sum_{i \in I} \alpha_q(\sigma_i^{Q+} + \sigma_i^{Q-}) + \\ &\sum_{f \in F} \alpha_{ef}(\sigma_f^s) + \sum_{e \in E} \alpha_{ef}(\sigma_e^s) + \alpha_\sigma \sigma_A \end{aligned} \tag{4-36}$$

Where

α_p = Penalization function for P
α_q = Penalization function for Q
α_{ef} = Penalization function for power flow in lines and transformers
α_σ = Penalization function for area power balance

Penalization cost functions are accordingly:

$$\alpha(x) = \begin{cases} k_1|x| & \text{if } 0 \leq |x| \leq x_1 \\ k_2|x| & \text{if } x_1 < |x| < x_2 \\ k_3|x| & \text{if } |x| \geq x_2 \end{cases} \tag{4-37}$$

The optimization problem can finally be summarized in vector form as:

$$\begin{aligned} \underset{x_c,\, u_0}{\text{minimize}} \quad & f(x_0, u_0) \\ \text{subject to} \quad & g(x_c, u_0) = 0, , \\ & h(x_c, u_0) \leq B(m), \end{aligned} \tag{4-38}$$

In order to increase the solution speed, it is required to use optimization solvers that are efficient and handle high-dimensional problems. In [150] a solution strategy is proposed based on the use of IPOPT. In [151], [152] and [153], IPOPT is shown as a high-performance algorithm for solving power system problems that includes OPF, SCOPF, and state estimation.

4.3 Methodology

The solution process initially includes the SCOPF decomposition into subproblems that include a master problem and other subproblems (contingency states). The first stage includes solving the base case OPF. Since the problem is non-convex, different initialization points were used to verify that the optimum point achieved coincides under different initial points. The second stage includes the modification of the limits of the existing constraints according to the results in the contingency states.

An algorithm based on existing tools in Matpower [154] and IPOPT [150, 155] is used to solve the SCOPF described in Section 4.2 [156, 157]. The algorithm consisted of data processing stages for the input and output of information and data results. The other stages were classified into three different groups: Parallel OPF, contingencies and constraints update.

Figure **4-1** shows the iterative process for updating constraint limits and solving the SCOPF problem. The pre and post processing stages include blocks for handling shunt reactive elements, which presented discrete reactive compensation blocks. These stages are found in the *SwShunts2Gen* and *SwGen2Shunts* blocks (see Figure **4-1**). With these blocks it is possible to include shunt compensation devices using a continuous model. Continuos functions allow OPF solution using the IPOPT. The other stages parallel OPF, constraint handling and contingencies, are described in depth in later sections. For algorithm stop, the following rules were included: penalty cost after new iteration does not have a considerable cost reduction or a maximum number of iterations is reached.

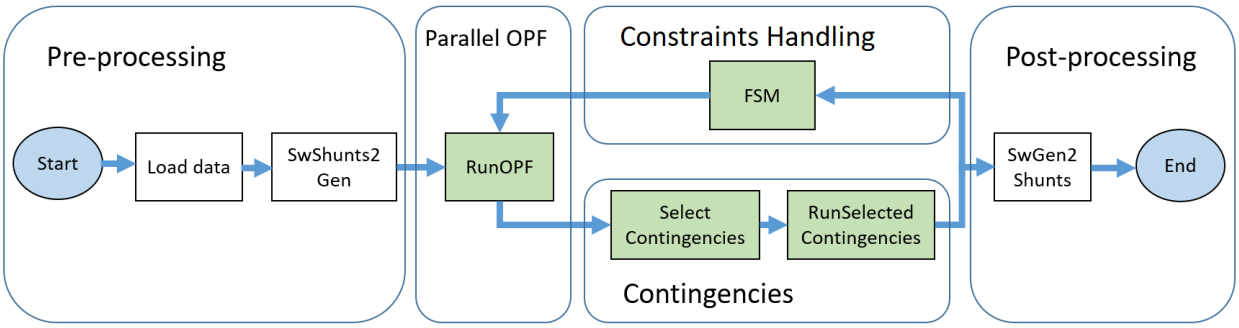

Figure 4-1: Overview of sections composing the main algorithm

4.3.1 Parallel OPF (Optimal Power Flow)

The *RunOPF* routine runs in parallel as shown in Figure **4-2**. The *load parallel seeds* function creates different initialization points to be solved using the IPOPT. Different solvers used different solution variables for initialization as well as different tolerance levels to verify solution times and optimal points reached. The linear solvers used were Multifrontal Massively Parallel sparse direct Solver (MUMPS cite AGULLO2008, MA57 cite Duff2004, or MA86 cite Hogg2010AnIS).

Different optimal solutions can be reached according to the selected algorithm and the defined tolerance. Once convergence is found, the optimum reached is verified. If the solution is less than the initial one, the time is increased and the tolerance value reduced. Once all the seeds have finished or the maximum time threshold reached, the best optimum is chosen as the starting point of all the master problem and sub-problems. The next stage is the *screening contingencies*.

4.3.2 Contingencies

4.3.2.1 Contingency Screening and Ranking

Since the number of contingencies in a large power system is considerable, contingency ranking and screening is used for medium and large-scale networks (e.g. more than 1,000 buses). The methodology for that selection comprised a set of sorted lists that reflected the likelihood for each contingency to develop overloads, voltage violations, and non-convergences. The strategy considers branches and generators independently, as shown in Figure **4-3**.

The main features for contingency screening are summarized in Figure **4-3** and described in the criteria below:

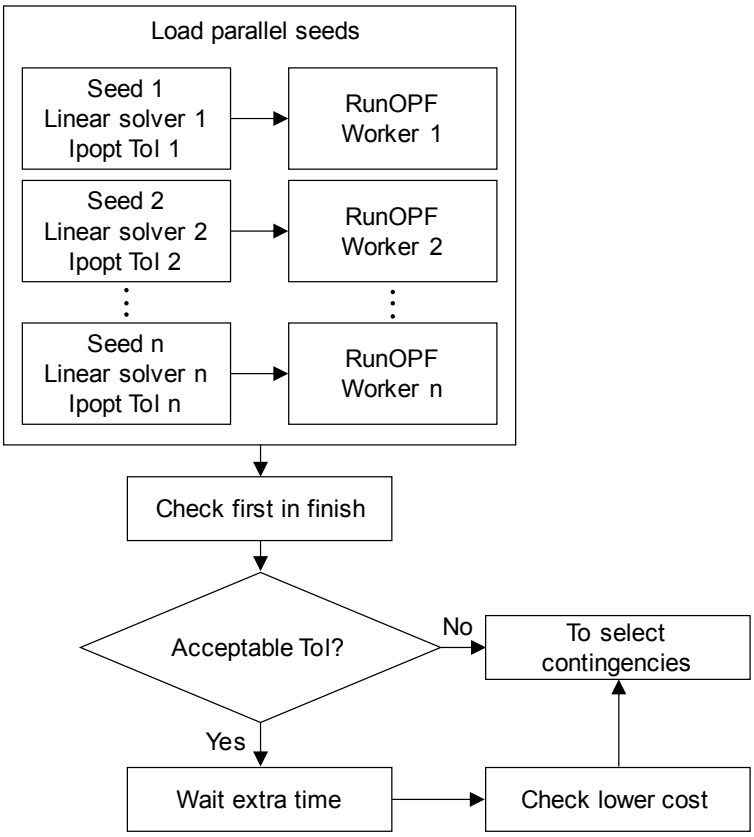

Figure 4-2: Parallel optimal power flow.

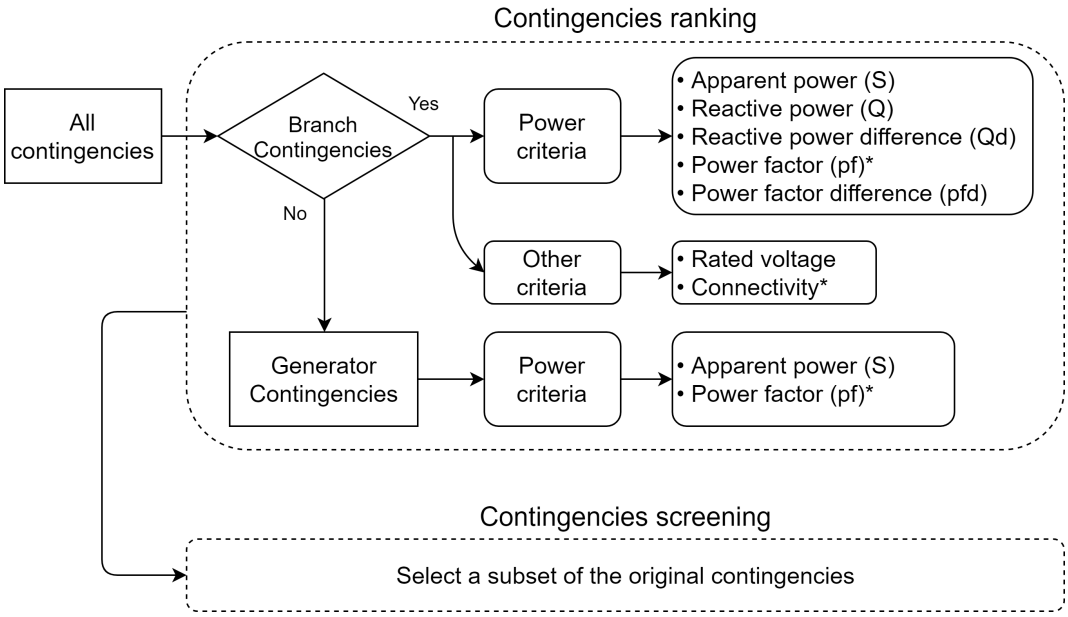

Figure 4-3: Contingencies selection flowchart

- *Apparent Power (S)*: Apply to both branch and generator contingencies.
- *Reactive Power (Q)*: Apply to branch contingencies.
- *Reactive Power Difference (Qd)*: Apply to computed branch and generator contingencies as Q difference between *origin* and *destiny* buses.
- *Power factor (pf)*: computed with Equation 4-39 for both branch and generator contingencies.

$$pf = cos\left(atan\left(\frac{Q}{P}\right)\right) \tag{4-39}$$

  Where:
  - pf: Power factor
  - $atan$: Four-quadrant inverse tangent function
  - P, Q: Active and Reactive power
- *Power factor difference (pfd)*: Apply to reactive power difference. *pfd* is a computed difference between *from* and *to* buses.
- *Rated voltage*: This criteria sorted branch contingencies according to the highest rated voltage between *origin* and *destiny* buses.
- *Connectivity*: Number of links among buses. Higher number of links represent higher stress in the system when a link is out of service.

Based on Figure **4-3**, contingency screening is performed for branch outputs and generator outputs. 7 criteria were used for line or transformer outputs, 2 criteria for generator outputs. Most of the lists were sorted in descending order, except for *Power factor* and *Connectivity* which were sorted in ascending order and marked with (*).

Once all lists were sorted according to the aforementioned criteria, a unique list of contingencies were computed from combination of those criteria. Each branch and generator were respectively ranked on each of the 7 and 2 lists depicted in Figure **4-3**. Given this, all the positions a single branch or generator occupied in the different lists were averaged to create a unique list of sorted contingencies for branches and generators, respectively. From these two lists, one for branches and one for generators, the top x contingencies were selected to continue as the input for *Contingency evaluation* stage.

4.3.2.2 Contingency Evaluation

In order to increase the solution speed, multiple computer cores are used to solve multiple contingencies in parallel. The algorithms shown in Figure **4-4** depict the calculation process in parallel. Once the contingency appears, the power flow is solved using a Newton Raphson method with the same generation and load conditions as the base case.

If there are contingencies with large power variations in the pre-post contingency state, considerable differences are stored in the variables ΔP (Equation 4-22). These conditions can appear when a line transfers large power flows or generation in an area is lost. When this situation appears in the problem, an adjustment in generators active power of the area is carried out. The new generator set point is carried out according to the participation factors of the generators in the area. Those generators that reach active power limits are fixed to their maximum limits, the remaining ΔP is distributed among the other existing units. The process is repeated iteratively until all the power difference is completely covered by the existing units, using the participation factors of each generator.

Some additional conditions that may arise include differences in active power between the programmed active power and real power flow results due to QV violations in certain generators (photovoltaic). In such cases, the PV buses were switched to PQ in order to satisfy Equation 4-23. These modifications can generate changes in the voltage magnitude of the PQ buses as well as in the power network losses. Changes in the power losses result in active power differences that can be observed in the slack variables. These conditions require new iterations of the algorithm in order to re-dispatch the existing units. New generation re-dispatches reduces ΔP differences. The process is repeated until the minimum deviation threshold is reached.

If the algorithm does not achieve solution from the units re-dispatch, a new *Noconv State* is created as described in the finite state machine in Section 4.3.3.1.

4.3.3 Constraint Handling rules

4.3.3.1 Finite State Machine

The stage of Finite State Machine (FSM), is created to update the base case constraint limits according to the non-convergence cases. The stage updates soft and hard limits for branches overloads and bus voltages violations identified in the *run selected Contingencies Stage* in Figure **4-1**.

Once the set of relevant contingencies in Section 4.3.2.1 are selected and run, violations are

collected and summarized in the categories: overloads, voltage violations, and non-convergence. Once all the previous information is organized and listed, the FSM updates the inequality constraints in the OPF base case problem prior next iteration.

Figure **4-5** shows the different states and transitions of the FSM stage. The FSM states are described in Section 4.3.3.2.

4.3.3.2 Updating Constraint Limits

The constraints limits were tuned based on FSM states, Figure **4-5**. The rules in the FSM update the $B(m)$ term in Equation 4-38.

When a violation appears on a line or transformer, an *overloads state* is activated as shown in Figure **4-6**. The results update maximum loading branch limits.

When a bus voltage violation appears, rules in Figure **4-7** are applied. In this case, the rule is updated base on Equation 4-40.

$$\begin{aligned}((v_k - v_{min} < 0.125)||(v_{max} - v_k < 0.125)) \\ \& \, |v_{base} - v_{lim}| < 0.01\end{aligned} \tag{4-40}$$

Power factor in the destination or origin buses was included as another criteria for voltage update limits. Low power factors show reactive power requirements and end up in low voltages in the destination bus and overvoltages in the destination bus once an outage appears on the branch.

Once marginal voltage or power factor criteria is not met, contingencies are updated in the base case as follows:

$$q_f^d < q_{f_{max}} \tag{4-41}$$

$$q_f^o < q_{f_{max}} \tag{4-42}$$

$$q_e^d < q_{e_{max}} \tag{4-43}$$

$$q_e^o < q_{e_{max}} \tag{4-44}$$

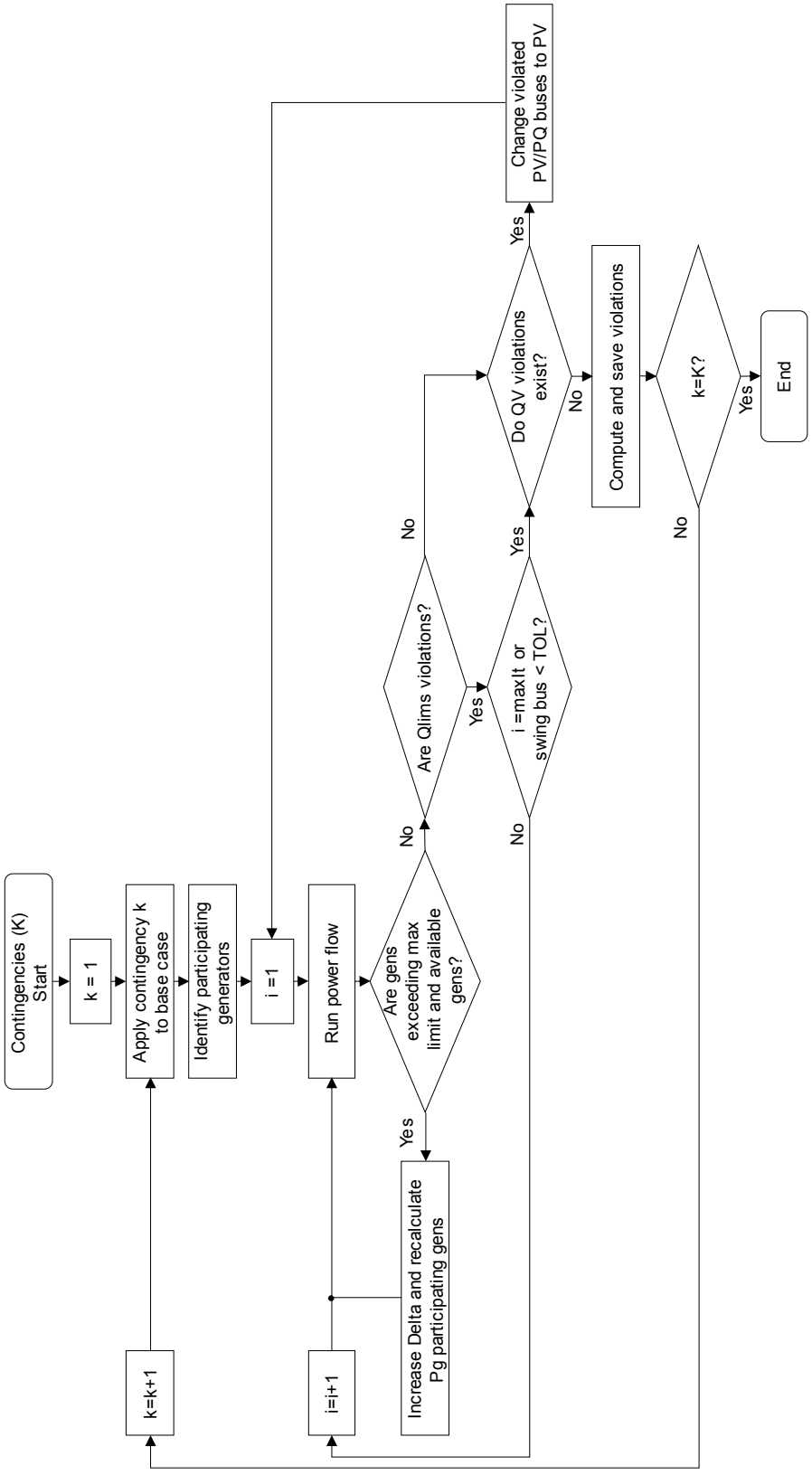

Figure 4-4: Active power re-dispatch and PV/PQ switch algorithm

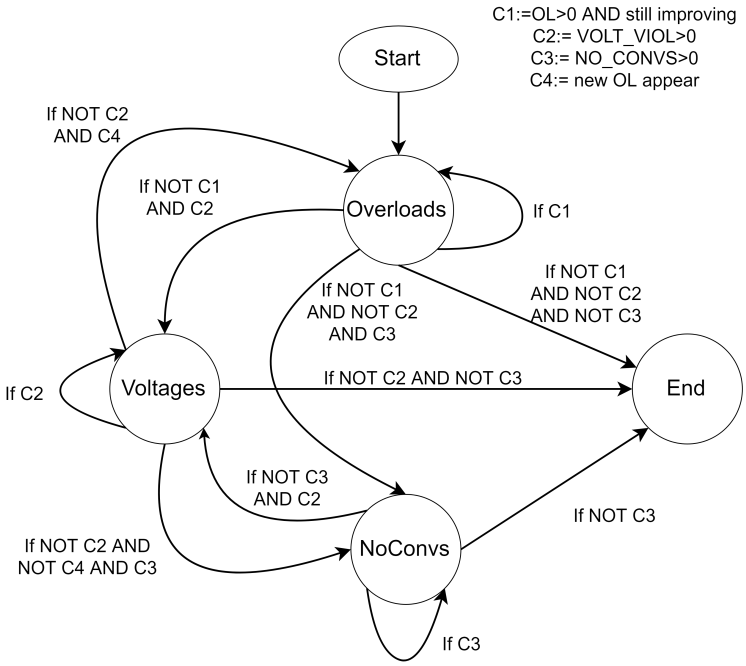

Figure 4-5: FSM transitions

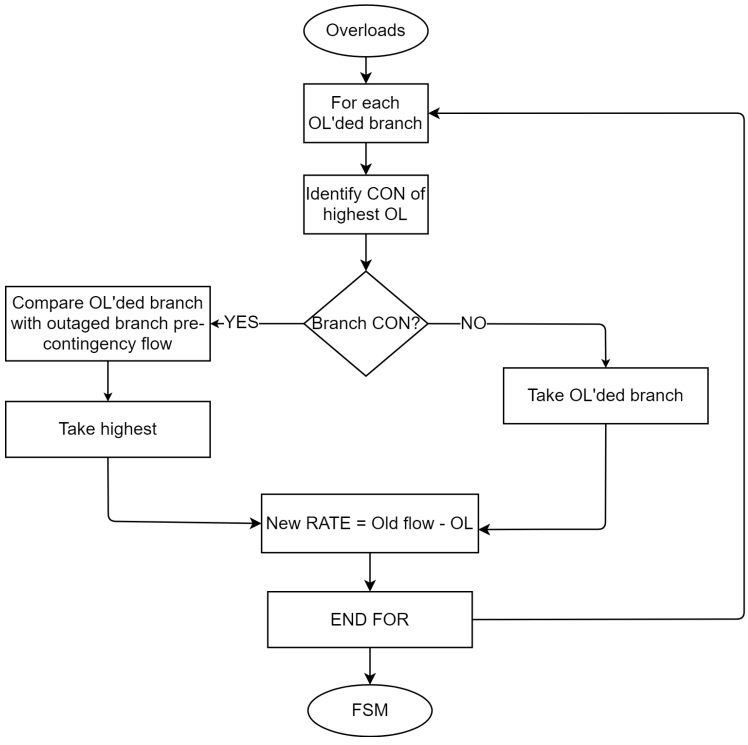

Figure 4-6: Branch limits updating

When maximum and minimum voltage limits violations appear, limits are updated as shown in Figure **4-7**.

Finally, if the state is *non-convergence*, the maximum generation limits are updated, as well as the branch limits modified. The limits new limits follow the following updates $\overline{P_g}(25\%), \overline{R}_e(10\%), \overline{s}_f(10\%)$.

4.4 Results

The SCOPF problem mentioned in Section 4.2 is solved with the algorithm proposed in Section 4.3. The algorithm was implemented in Matpower Toolbox [154] using an IPOPT solver [150, 155]. The set of evaluated networks are shown in Table **5-3**. These networks were used in the *ARPA-E Grid Optimization Competition - Challenge 1* [157]. The same Table **5-3** shows the results for the proposed algorithm. All networks were solved using a 64-bit Linux distribution of Matlab® 2019, Intel(R) Xeon(R) CPU E5-2680 @ 2.70GHz, 128 GB RAM memory and 16 cores.

To achieve the optimal solution of the base case, a total of 16 seeds were evaluated using a parallel distribution in 16 computer cores in order to optimize the systems under study. The evaluation of multiple initial points can allow different optimal solution [127]. The 16 seeds included different combinations, modifying the following fields:

- Linear Solver: mumps, ma57 [150].
- Strategy: monotone (default), adaptive [150].
- Oracle: quality function (default), Loqo [150].
- Seed: initialization of decision variables from a base case (warm starting), from "zero" condition (cold starting) or from previous algorithm iterations [158],
- Initial voltage: from a base case (warm starting) or set to 1 p.u.

Initially, Contingency Screening (CS) was tested. Each network in Table **5-3** was tested with different percentages of the total number of contingencies (25%, 50%, 75%, and 100%). The percentages were applied to generation, line and transformer contingencies.

To identify the performance of the algorithm, the evaluation of 100% of the number of contingencies was initially carried out to identify the total of violations resulting from overloads and under- and over-voltages, the non-convergences of the contingency scenarios

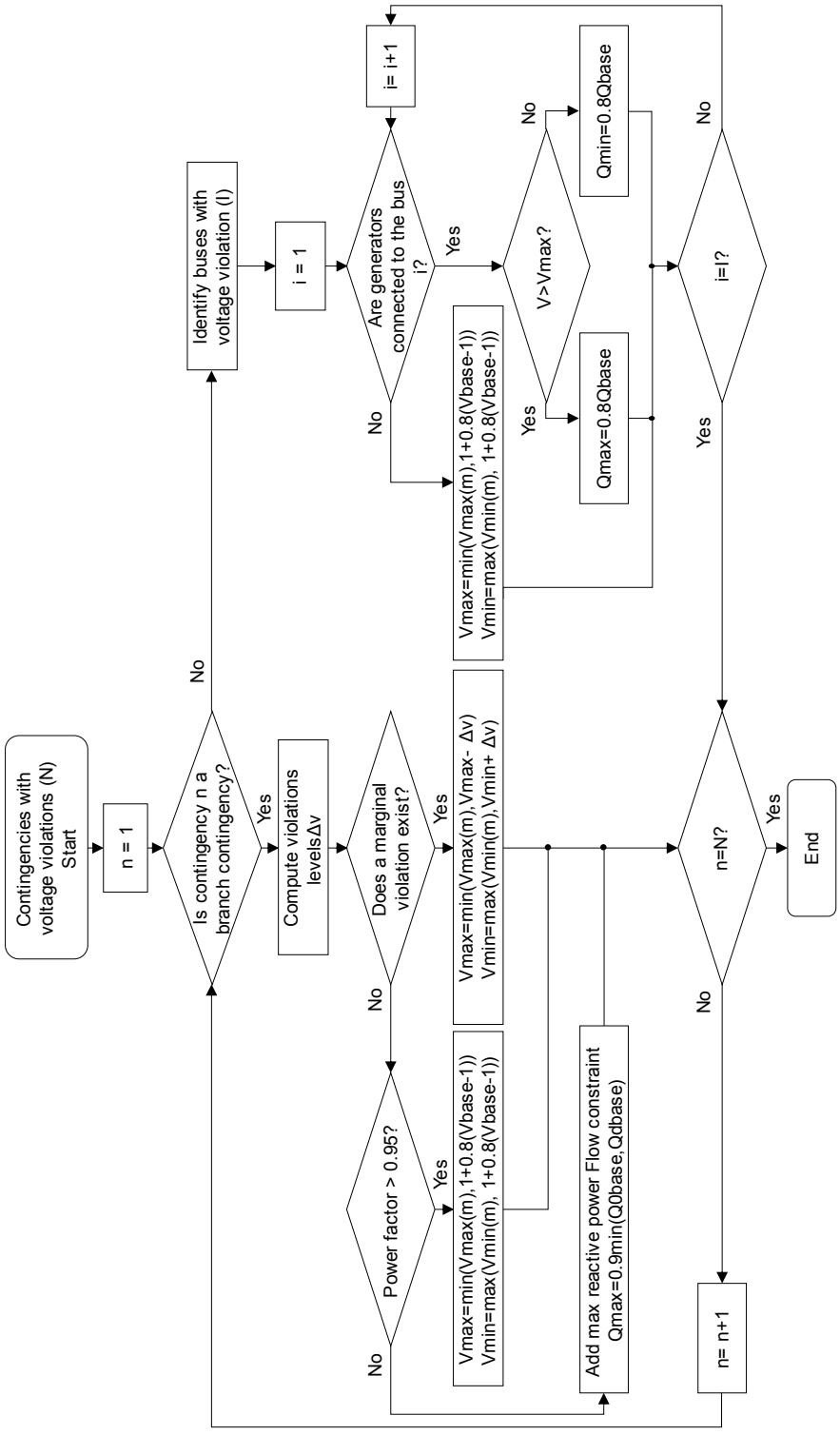

Figure 4-7: Voltage limits updating

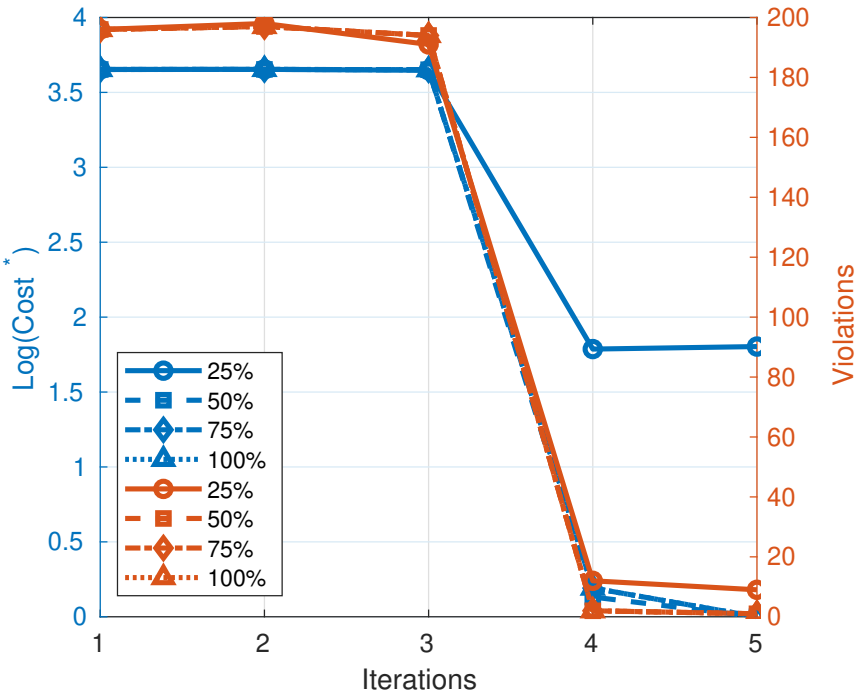

Figure 4-8: Cost and violations in function of iterations for network 1 for different percentage of CS

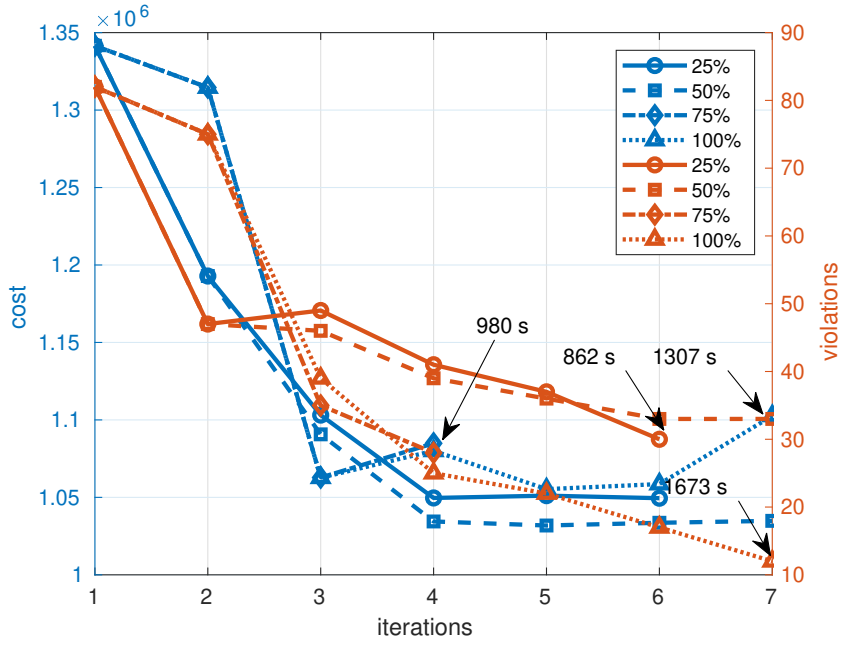

Figure 4-9: Cost and violations in function of iterations for Network 2 for different percentage of CS

Table 4-1: Description of networks tested

Net.	Buses	Generators	Loads	Branches	Transformers	Cont.	Shunts	Areas
1	500	224	281	540	193	786	44	1
2	4,918	1,340	3,070	4,412	2,315	5,085	732	31
3	11,615	899	19,272	13,967	5,936	8,747	1,332	1

were also identified. The total operating cost included penalty cost for violations of soft and hard constraints and nodal imbalances. The voltage violations not solved by the FSM, are solved modifying the nodal balances, according to Equations 4-33 and 4-34. The penalty cost in Equation 4-37 used $k_1 = 1000$, $k_2 = 5000$, $k_3 = 10e6$, $x_1 = x_2 = 2$, and $x_3 = 50$.

Figure **4-8** shows the cost achieved during each iteration¡ similarly, shows the number of existing violations for network 1. The results of the algorithm are shown under different percentages in the CS process. Figure **4-8** shows convergence of the SCOPF algorithm in 5 iterations. The minimum cost achieved was $ 2.63e5. The tests showed a similar cost when only 50% CS is used.

$$Cost^* = log_{10}\left(\frac{Cost}{C_{min}}\right) \tag{4-45}$$

Figure 6-8 shows non significant time differences in medium networks such as Network 1. In this network, total solution time using 25% contingencies is 33.08 s, while using 100% contingencies resulted in 42 s. This shows that an increase of 75% of the contingency number, results in 25% of time increase.

Figure **4-9** shows the results for a large power network like Network 2. In this case the minimum cost is similar (with a difference of less than 2%) using 50% and 75% CS. The algorithm shows convergence in 5 iterations for different percentages of CS. On the other hand, the number of final violations due to bus imbalances is 33 using 50% CS and 23 when using 100% CS. In terms of time, the algorithm with 25% finished in (682 s) and the final cost was lower than for the system using 100% CS. Using 75% CS, convergence of the SCOPF algorithm was achieved in just 4 iterations, however, the total time was greater than 25% (980 s). Results using only 25% are shown aligned with real-time applications.

The results of another large network are shown in Figure **4-10**. In this case, network 3 reaches convergence using 50%, 75% and 100% of CS. With 25%, the penalty was high, being 64 times higher than the minimum cost reached with higher percentages of

contingency screening.

Table **4-2** summarizes the results for Network 3. In that system, a very low number of contingencies results in a high number of violations. Although the number of violations to the voltage constraints is low (0.18%) with respect to the total number of buses, the final cost is unacceptable. The cost achieved with 50%, 75% and 100% CS is acceptable. In terms of cost and time, the best performance was achieved using 75% CS.

Figure **4-11** shows the average contingency time for different newtwork sizes using a cluster of 16 and 72 cores. The difference in time for network 3 shows 52ms/cont. Thus, a 25% reduction in total contingencies would decrease 341 s per iteration. In that way 4 iterations of the algorithm would require approximately 1,178 s (19.6 min). This time is less than half the total time using 100% CS of 2.542 s.

Compared to other proposals, mainly that of Bender Decomposition in [159], the proposed methodology turns out to be faster. In [159] the average contingency time is 750 ms, while a network of 2,312 buses using this methodology presents times less than 50 ms when executed in 16 and 72 parallel cores.

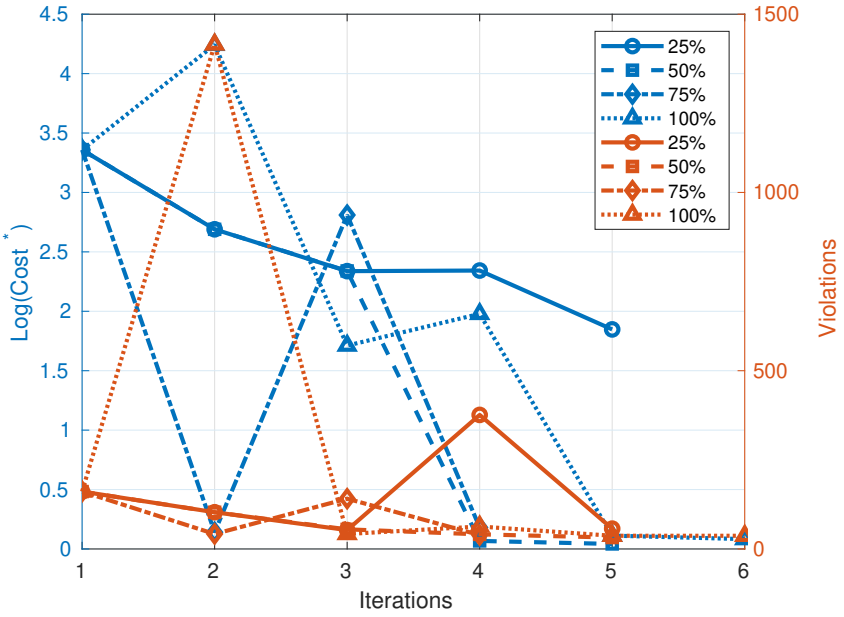

Figure 4-10: Cost and violations in function of iterations for Network 3 for different percentage of CS

Table 4-2: Summary Results for different percentage of CS

Screening percentage	25	50	75	100
Convergence Time [s]	2200	2817	2542	5352
Cost (1e8) [$]	1.3467	0.0211	0.0264	0.0231
Violations	22	2	0	2

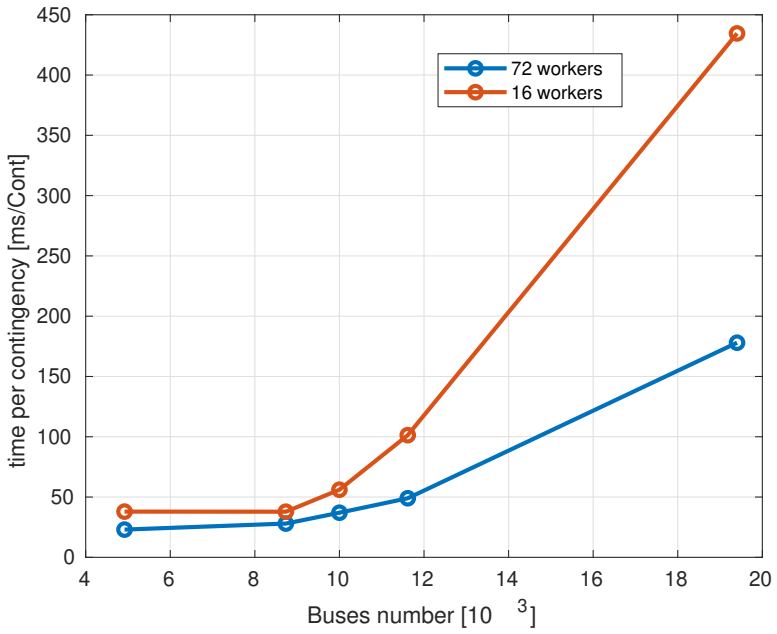

Figure 4-11: Average time per contingency in function of number of buses

4.5 Discussion

The CS process increases its effectiveness in large power networks. This is supported by the medium-sized network such as Network 1 (500 buses). This Network does not show positive results when contingency filtering is included. In that case times are not significantly reduced by including this strategy.

The number of cores available is another high-impact factor for fast contingency screening. This is one of the main processes of the algorithm. For online applications, the percentage of contingencies selected is related to the available time for algorithm solution, system size and the number of available cores. The criteria in Figure **4-3** are applied once the desired percentage of contingencies in the algorithm is known.

Traditional strategies rely on the use of SCOPF decomposition using multilevel optimization [126]. The proposed algorithm is based on power flow solutions. This strategy makes the algorithm easier for implementation with faster solution times.

4.6 Conclusions

This chapter proposes a SCOPF algorithm using modifications to a base OPF. The method shows practical applications for medium and large size systems, including networks with dimensions greater than those commonly analyzed in the literature. Parallel computing supported not only the search for initial optimum but also for contingency simultaneous solutions.

The strategy used is based on the power flow solution, which allows fast solutions compared to other strategies such as BD. This procedure speeds up troubleshooting time, making it effective for large power networks.

The CS algorithm showed adequate performance when filtering greater than 50% contingencies, this shows a suitable strategy for on-line SCOPF. Other vectorized and High Performance Computing (HPC) structures can offer a possible method for algorithm acceleration as shown in Chapter 3.

5 SCOPF using an PHC Strategy for Medium and Large Size Power Systems

5.1 Introduction

Given the greater dependence on electricity by our society, a high priority has been given to the continuity of electricity service. For this reason, multiple network codes have chosen not only to include in their demands requirements related to operation at the lowest cost, but also to safe operation. In this way a balance is struck between optimal and safe operation. These two conditions are covered by formulations such as Security Constrained Optimal Power Flow (SCOPF) [117–121]. The Department of Energy (DOE) estimates that billions of dollars could be saved by optimizing the operation of the system and ensuring that the system can operate in the event of an N-1 contingency. For this reason the energy industry has turned its attention to this type of development. However, some limitations do not make this exercise an easy task. Among them is the non-linear and non-convex nature of the problem, which together with the integration of a large continuous and integer decision variables make it fit into the category of complex mixed optimization problems [119, 120, 122, 123].

There have been different proposals for solving this type of problem. Some include linearization, convexification, decomposition, decomposition and contingency screening, among others [27, 160]. These strategies have had responses made in small and medium networks, but limited applications have appeared in large networks as shown in Chapter 4. Reducing its application in real networks that have thousands of buses and contingencies.

One of the alternatives to reduce the solution time of the SCOPF problem is the use of non-traditional structures such as Parallel and Heterogeneous Computing (PHC). The reference [161] shows accelerations of up to 56 times the normal computation time used to solve an OPF problem. It does this by using CPU and GPU structures. In this case, the algorithms are applied to 300 bus networks. The authors in [162] compare different solution strategies based on CPU and GPU frameworks. In that approach, a power flow problem is solved for a system with up to 3,012 buses using a CPU-GPU structure.

Reference [41] shows a compensation algorithm for asynchronous parallelization, which allows solving large and sparse networks with a large number of contingencies. The algorithm is tested on networks of up to 13,659 buses.

Although these PHC architectures have been applied using CPU and GPU to electrical systems, the majority have been carried out in power flow, transient stability, contingency evaluation, smart grids, among others. Section 2.3 shows limited applications in the solution of SCOPF and OPF on large power networks using CPU-GPU architectures. This section accelerates the execution of the SCOPF algorithm based on the handling of constraints from Chapter 4, using an PHC architecture based on CPU-GPU. The main objective of the acceleration process is to meet the required operation times with structures that do not involve robust and high-cost networks. The algorithm includes the complementary constraints from Chapter 4 such as spin reserve and AGC, contingency screening, generation re-dispatch, and PV/PQ switching.

This chapter applies a PHC architecture in medium, large and complex power grids, to solve SCOPF problem. The algorithm includes:

- Acceleration of the Parallel Computing (PC) strategy in Chapter 4 using a PHC architecture that optimizes data management using CPU and GPU to solve SCOPF problems in medium, large and complex power systems.
- Application of integrated sparse matrix algorithms in SCOPF to solve medium and large power grids.

The rest of the chapter is organized as follows. Section 5.2 presents a summarized formulation of the SCOPF problem and explains the propose strategy to solve it. Section 5.3 present the PHC architectures tested in the research. Section 5.4 make a list of the study tests and conditions to evaluate PHC architecture performance. Section 5.5 shows the results for benchmark networks and PHC architectures. Finally, section 5.6 presents conclusions and future work in the area of SCOPF using PHC in large and complex power grids.

5.2 Mathematical Formulation

One formulation for SCOPF is shown in [156]. The problem is represented with a bi-level formulation. A base case for the normal state of operation, denoted $k = 0$. $N - 1$ scenarios for K contingency cases, represented by indices k in the set $SC = 1, \ldots, K$. Contingencies represent faults in generators, lines, and transformers. The decision variables are divided into two groups: The base case includes voltage magnitudes and angles, real and reactive

power in generators and controllable reactive shunt devices. The contingency cases in K include the variables of the normal operation case, together with the AGC variables or area power balance variables, represented by ΔP^{sc}. The number of decision variables is denoted by n_{xy}. The vectors of the primary variables are represented by $x_k \in \mathbb{R}^{n_{xy}}$ for $k = \{0\} \cup SC$. Soft constraints are included in order to have workable solutions in the SCOPF formulation [156]. Slack variables are represented by $\sigma^k \in \mathbb{R}^{n_{\sigma_k}}$ where n_{s_k}. Nodal balance variables are represented by $\sigma^{k,+}$ and $\sigma^{k,-}$ and lines and transformers by $\sigma^{k,s}$.
The SCOPF formulation is expressed as below:

$$\underset{x_0, \sigma^0, x_k, \sigma^k}{\text{minimize}} \quad c(x_0) + c_0(\sigma^0) + \frac{1}{|K|} \sum_{k \in SC} c_k(\sigma^k) \tag{5-1a}$$

subject to

$$f_0(x_0) = \sigma^{0,+} - \sigma^{0,-}, \tag{5-1b}$$

$$g_0(x_0) \leq \sigma^{0,s}, \tag{5-1c}$$

$$f_k(x_k) = \sigma^{k,+} - \sigma^{k,-} \quad k \in SC, \tag{5-1d}$$

$$g_k(x_k) \leq \sigma^{k,s} \quad k \in SC, \tag{5-1e}$$

$$h_k(x_0^{p,q,v}, x_k^{p,q,v,\Delta P^k}) = 0 \quad k \in SC, \tag{5-1f}$$

$$x_k \in \chi_x, \sigma^k \in \Sigma_k, k \in \{0\} \cup SC \tag{5-1g}$$

The objective in (5-1a) includes the costs associated with generation operation and violations in nodal imbalances or overloads in transformers and lines under normal and contingency states. Nodal balance and branch overload constraints are represented in (5-1b) and (5-1c). In the case of contingency, they are represented by (5-1d) and (5-1e). The contingency generation constraints are shown in (5-1f). The superscripts p, q are associated with injections of active and reactive power from generators, v is associated with objective generator voltages. ΔP^k represents the power fit between base and contingency cases. The last constraint (5-1g) is associated with χ_x and Σ_k extended real coordinate space, respectively $\mathbb{R}^{n_{xy}}$ and $\mathbb{R}^{n_{\sigma_k}}$. Therefore, the constraints on (5-1g) include upper and lower bounds on the slack and decision variables. The complete formulation can be seen in Chapter 4 and [40].

The challenges of this problem are reduced in: Non-linearity and non-convexity for nodal balances (5-1b) and (5-1d). The non-linear, non-convex and non-differentiable responses (5-1f). The constraint (5-1f) includes the addition of binary variables or iterative strategies as in Chapter 4. Another relevant factor is related to the systems size, which increase the complexity of the base case solution and contingencies number. The network includes thousands of buses and branches in the system that are close to real networks as shown in [156]. Constraints in (5-1b) to (5-1g) limit easy parallelization.

The proposed methodology follows the algorithm in Chapter 4. The methodology solves a bi-level problem to solve a base case and contingencies. The base case solution includes the definition of generation, line and transformer values in order to obtain the lowest system cost. Subsequently, it updates the constraint limits for each of the existing elements in order to satisfy the maximum allowed in case of faults and out of service conditions for the existing elements. In other words, the methodology does not include new contingencies for the base case. Solving the SCOPF problem not only guarantees an optimal solution of the base case but also close to optimal operations in contingency state.

Figure **5-1** shows a summary of the algorithm used in Chapter 4 for the solution of SCOPF. The methodology breaks down the problem into a base case and a study case. The first stage in blue dotted lines includes the solution of the OPF subject to normal operating constraints. The second stage in green lines, include constraint handling using an update of the existing limits through a finite state machine. In this way the steady state constraints are updated (5-1b)-(5-1c). The process stops under three conditions: a) The penalty cost is less than a defined percentage of the objective function b) A number of iterations is reached c) A time limit is reached. The full approach is described in Chapter 4.

Since the real-time responses to the SCOPF problem can be limited due to the processing times required when solving large networks, different strategies have been applied by different authors. These include contingency screening, power flow simplifications and parallel computing.

The contingency screening process and flow assessments during contingency states are included in [163]. The contingency screening is carried out for both generation and line outputs. The generator outputs are organized using apparent power and power factor measurements. Contingencies in lines and transformers use the variables of load flow, voltage levels and power factor. The power flows give preliminary information of stress system conditions that can generate greater impacts on the network. OPF runs are not performed in the contingency screening stages.

In order to accelerate processes with a long execution time such as the contingency assessment process, an architecture using GPU and CPU is applied in this chapter. The use of mixed computing environments enables efficient solutions for large and complex networks. A detailed description of the parallel structure is given in Chapter 2 and section 5.3.

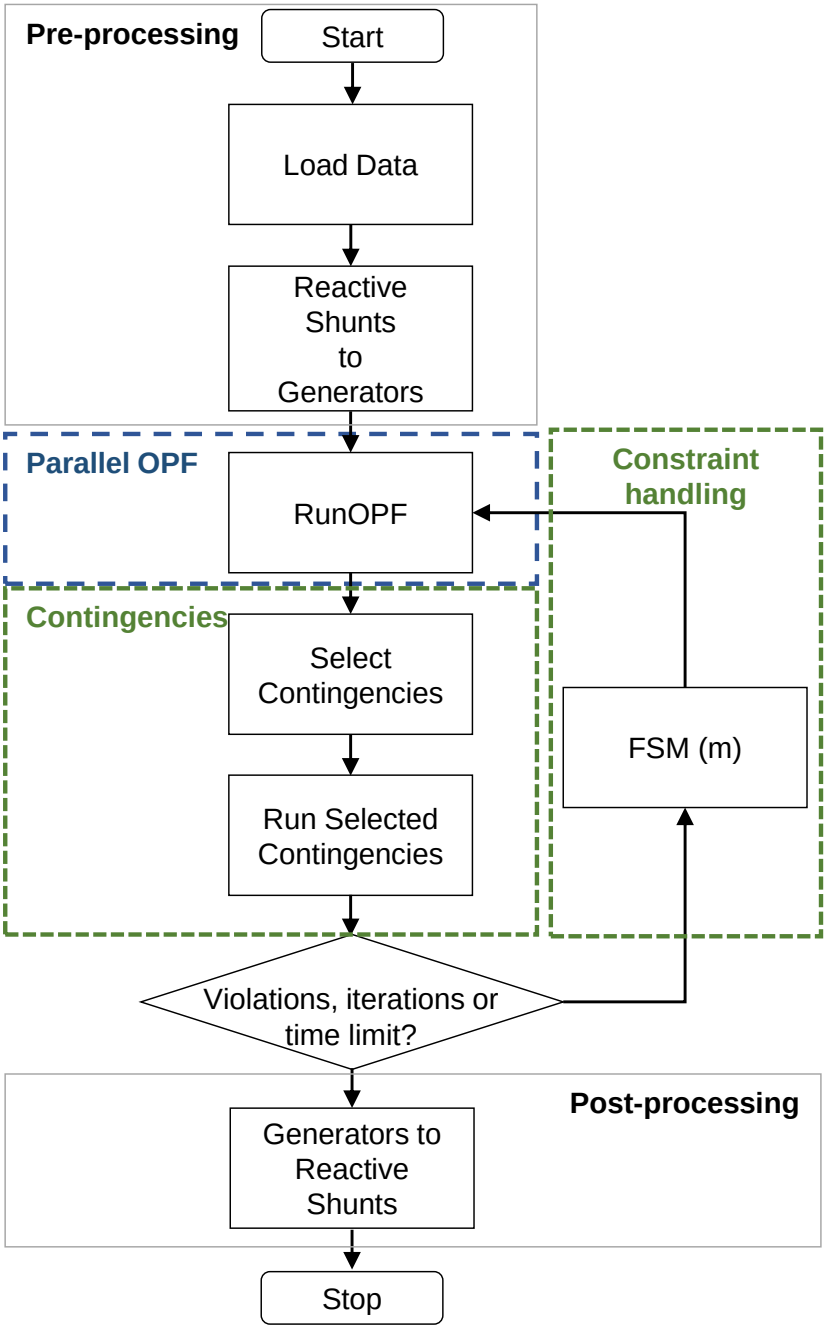

Figure 5-1: Flow chart of fast SCOPF algorithm

Table 5-1: Pseudo code for SCOPF algorithm (Average Time)

Pseudo Code	**Time (s)**
Pre-processing	19.69
Run OPF	732.99
Select Contingencies	823.44
Run Contingencies	16,276.86
Constraint handling	0.29
Post-processing	1.51
Total Time	**17,854.8**

5.3 Heterogeneous and Parallel Implementation CPU-GPU

To reduce the solution time of the SCOPF algorithm, an PHC architecture is proposed. The following sections describe the proposed architecture and the GPU and CPU architectures.

5.3.1 PHC Architecture

The solution of large and complex networks has today become one of the realities that control centers and system operators must face. The strategies shown in [119, 163] appear as possible solutions to ensure quick responses while simultaneously achieving optimal and secure operation.

Table **5-1** shows the processing times required to solve a SCOPF problem using a single core of a 4,918 bus network using a single computer core. As described in the table, the most time consuming process is contingency resolution. This process is presented as the bottleneck of the algorithm.

Based on [24, 161], the use of thousands of threads can be achieved from vectorized data structures. For these solutions to be economical, the use of local CPUs and GPU threads is done. The acceleration of the algorithm in this section, Figure **5-1**, is done using dozens of GPU cores to reduce execution time during contingency assessment. All other activities are kept supported using CPU structures.

To accelerate the execution of the SCOPF algorithm, the step that is a candidate to be accelerated is the contingency evaluation, not only because it is the most time consuming, but also the one that presents an adequate structure to be parallelized. According to the algorithm in Figure **5-1**, contingency parallelization is used to update the constraints of

the base case optimization problem, as shown in the equations (5-1b) - (5-1c). Some strategies suggest the use of cloud services, use of CPU parallelization and even various GPU strategies [84, 85, 164–175]. The implementation from parallelizations that include CPUs is one of the most tentative; however, the cost of thousands of cores and the complexity of creating multi-node clusters, shows it as a not-so-easy-to-implement solution.

The general structure of the SCOPF algorithm is mainly programmed with Matlab using CPU cores. The activities of parallelization of activities using GPU through the CUDA language. In this architecture the objects mainly reside on the CPU and interact with the GPU through different data pointers. The host calls class methods and launches CUDA kernels that execute the operations on the device.

5.3.2 Network Data

Data entry is done through MATPOWER, which performs a reading of the available buses, connection branches, transformers and loads, as well as the operating costs of each of the generation units. This data is transferred to the device from the CPU that copies the data to the device using different pointers in order to create the admittance matrix, the Jacobian, the complex voltages, power arrays and the GPU unbalance vectors. The Jacobian matrix and the nodal imbalance matrix are copied to the host in order to proceed to solve with Solvers dedicated to this task. Table **5-2** shows the list of variables used in the power flow.

CPU keeps control of data stored on GPU saving the pointers in a single structure. This data also contains the CUDA Handles for use *cusparce* and *cublas* libraries functions and stream id for kernel concurrency.

5.3.3 Sparse Matrix

As large-power systems handle a large number of elements, these are stored following the Sparse Coordinate Format (COO) and the Compressed Sparse Row Format (CSR). For an array A of dimension nxn with nnz number of nonzero elements, the COO format uses a_i for row indices, a_j for column indices, and a_x array containing nonzero values from the matrix, this means $A[a_i[i], a_j[i]] = a_x[i]$. The CSR format also consists of tree arrays, two of those are a_j and a_x, the same as the COO format, the last a_p of length n + 1, storing the location of the first zeros of each row in a_j and a_x and the number of nonzero values in the i-th row is $a_p[i+1] - a_p[i]$. The COO matrix must be built before calculating a_p from the CSR matrix. The use of both formats does not require a significant number of

Table 5-2: Networks variables

Variable	**Data type**	**Stored in**
bus_type	int array	CPU/GPU
bus_type_index	int array	CPU/GPU
bus_pd	real array	CPU
bus_pq	real array	CPU
bus_ds	real array	CPU
bus_bs	real array	CPU
valV	complex array	CPU/GPU
valdx	real array	CPU/GPU
br_status	real array	CPU
br_f	int array	CPU
br_t	int array	CPU
br_r	real array	CPU
br_x	real array	CPU
br_b	real array	CPU
br_tap	real array	CPU
br_shift	real array	CPU
gen_gbuson	real array	CPU
gen_pgon	real array	CPU
gen_qgon	real array	CPU
Ybus	SparceMatrix	CPU/GPU
Jacobian	SparceMatrix	CPU/GPU
mismatch	real array	CPU/GPU
Sbus	complex array	GPU
Ibus	complex array	GPU

resources, however, it allows to increase the efficiency of the algorithm when it is executed.

5.3.4 Data Transfer CPU-GPU

CPU sends branch arrays to GPU (status, R, X, B, TAP, SHIFTF, F and T) GS and BS arrays from bus info and base MVA. GPU computes Y_{bus} matrix and returns it to CPU in COO format. After this step, GPU only keeps stored Y_{bus} matrix in both COO and CSR formats. Once previous process is complete, CPU sends set of generators arrays (GBUS, PG, QG) complementary BUS arrays (PD, QD, bus type, bus type index) and seed value for voltages array to GPU. GPU computes S_{bus} and keeps stored voltage array, bus type, bus type index and S_{bus}.

On Newton-Raphson loop, GPU sends mismatch vector and Jacobian matrix in COO format to CPU. After that, CPU solves the linear system. GPU gets and uses the voltage deltas array from CPU. In order to accelerated host-device communication and make possible data transmission and kernel execution, transmitted variables are stored in page-locked memory.

5.3.5 GPU Architecture

The GPU architecture is described in Figure **5-2** and Chapter 2. GPU are made up of Streaming Multiprocessors (SM) that contain different numbers of Streaming Processor (SP). Each SP containts arithmetic-logical units that allow calculations using floats and integers points. SMs share information using the global memory that uses a low latency channel. SMs provide kernels for the execution of different threads. The CPU and the GPU interact using a Express data bus - PCI. The developments shown in this Chapter were run on NVIDIA GeForce GTX 1060 card. This card has a GeForce GTX 1060 chip, 1280 SPs, 10 MPs, 98K shared Memory Per Multiprocessor, 65K 32 bit registers per MP and 6GB of DDR5 SDRAM. The GPU was programmed using CUDA language that allows execution of task in NVIDIA environments.

The CUDA execution architecture is shown in Figure **5-3**. Programs are started on the CPU host and sent to the device-GPU. Different kernels are sent simultaneously in thread blocks. Each programming thread is assigned to independent SPs. The records are executed individually until each process is closed. Nevertheless, threads share information within each memory block and not among them.

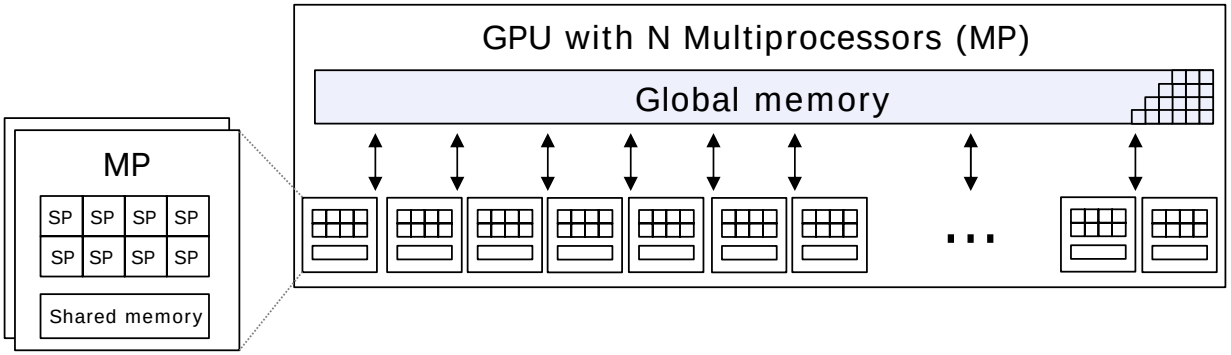

Figure 5-2: Typical architecture of GPUs

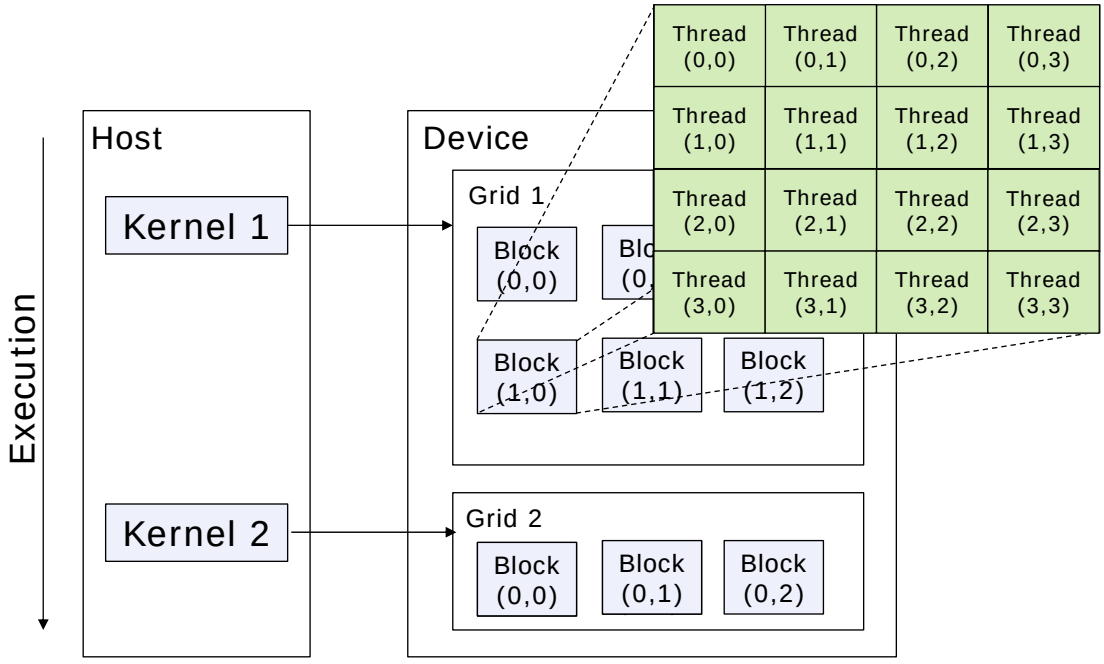

Figure 5-3: Execution model of a CUDA program

5.3.5.1 Client-Server Architecture

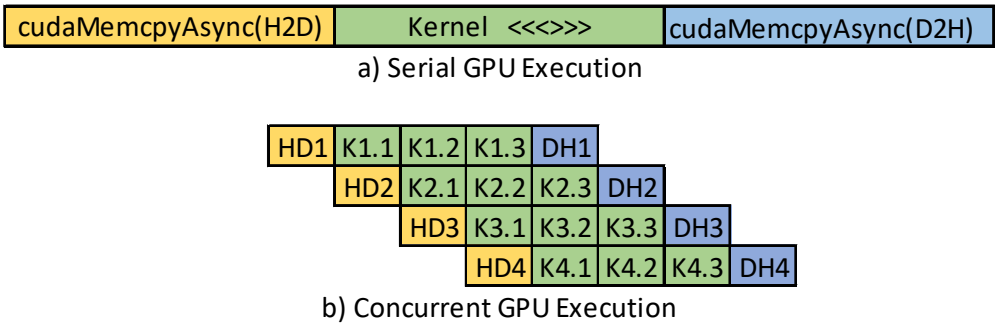

Figure 5-4: Client-Server architecture [4]

CUDA allows the execution of multiple kernels concurrently. To do this CUDA uses streams. These streams are defined as *"A sequence of operations that are executed in issue-order on the GPU"* [4]. Operations requested from different streams can be executed at the same time, if enough resources are available on the GPU. Multiple requests can be sent from the host, however, the request will be limited by the existing bottleneck in data transfer process. In this way only one problem can be transferred through the existing host-device channel. Figure **5-4** shows an example of a serial and parallel sequence. As the serial execution is observed, it includes a beginning and an end of a kernel. On the other hand, the execution of 4 kernels shows simultaneous execution, nevertheless, the execution does not start simultaneously.

5.3.5.2 Kernel Design Strategy

One kernel functions group can be executed N times in parallel if N different CUDA streams are launched. Since all scenarios are independent, if each scenarios have associated an stream, N scenarios can be computed in parallel. The proposed implementation assign a GPU stream to each available CPU core, an stream run all the steps of power flow from an scenario except for the linear solver that run on the CPU. In this way, each scenario can be solved asynchronously, but facing possible bottlenecks in communications, since there is only one channel between the GPU and the host. Even so, the possibility of running kernels while new kernels are being transferred will improve the performance of the algorithm. compared to do all the process in a single stream where the calculations and communication must be given serially.

Kernels were provided by self-built **cusparse** and **cublas** libraries. For cusparse and cublas functions, a handle is required, to this handles a stream id was set to keep concurrency.

5.3.5.3 Newton Raphson

One of the activities that requires the highest computational resources consumption is contingency evaluation. The process performs power flow evaluations after taking one or more elements in the system out of operation. For this process, only the fault of one element will be used. Different algorithms have been proposed for the solution, however, one of the ones that shows the highest convergence rate as well as precision is the Newton Raphson. The Newton Raphson algorithm is used in this opportunity. Algorithm 2 shows a partially GPU implementation. The stages in blue are those that have been parallelized. Initially the information is loaded into the CPU and transferred to the GPU using matrix notation. The processes are parallelized for the formation of the Jacobian matrix and the vector of nodal balances. At this point the elements J_{11}, J_{12}, J_{21} and J_{22} of the Jacobian matrix and P_i, Q_i, ΔP_i and ΔQ_i are calculated, from the vector of nodal balances. Once the arrays are created, they are sent to the CPU for solution. The convergence errors are calculated. If the nodal balance errors are not met, the process is repeated until convergence is reached in the algorithm.

Algorithm 2 Pseudo-Code for Parallel Power Flow Evaluations using Newton Raphson Method

procedure LF_NR_EC(Buses, Branches,P_{gen},Q_{gen},P_{load},Q_{load})
  $\mathbf{V_i}, \theta_\mathbf{i} \leftarrow [1], [0]$
  $\mathbf{Y_{bus}} \leftarrow YbusCalc\,(\mathbf{Branch})$
  while $\epsilon > \phi$ **do**
    $P_i \leftarrow P_i = \sum_{k=1}^{N} V_i V_k \left(G_{ik}\cos(\theta_{ik}) + B_{ik}\sin(\theta_{ik})\right)$
    $Q_i \leftarrow Q_i = \sum_{k=1}^{N} V_i V_k \left(G_{ik}\sin(\theta_{ik}) + B_{ik}\cos(\theta_{ik})\right)$
    $\Delta P_i, \Delta Q_i \leftarrow \Delta P_i = P_{G_i} - P_{L_i} - P_i, \Delta Q_i = Q_{G_i} - Q_{L_i} - Q_i$
    $\mathbf{J} \leftarrow f\,(\mathbf{Y_{bus}}, \mathbf{V}, \theta)$
    $\begin{bmatrix}\Delta\theta \\ \Delta\mathbf{V}\end{bmatrix} \leftarrow \left[\mathbf{J}\right]^{-1} \begin{bmatrix}\Delta\mathbf{P} \\ \Delta\mathbf{Q}\end{bmatrix}$
    $\theta_\mathbf{i}, \mathbf{V_i} \leftarrow \theta_\mathbf{i} + \Delta\theta_\mathbf{i}, \mathbf{V_i} + \Delta\mathbf{V_i}$
    $\epsilon = \max(\Delta P_i, \Delta Q_i)$
  end while
  return $\mathbf{V}, \theta$
end procedure

5.3.6 CPU Architecture

As shown in Table **5-1**, the activity with the highest consumption of algorithm processing time is the contingency processing. This process is accelerated using a GPU architecture.

Table 5-3: Description of networks tested

Network	Buses	Generators	Loads	Branches	Transformers	Contingencies	Shunts	Areas
1	500	224	281	540	193	800	44	1
2	4,918	1,340	3,070	4,412	2,315	5,074	729	31
3	11,612	899	19,272	13,967	5,936	8,747	1,332	1
4	19,402	973	12,928	22,950	11,754	13,391	2,451	1
5	30,000	3,526	10,648	32,020	3,373	10,810	1,275	16

The other stages are accelerated by CPU parallel implementations. Non-parallelized activities are those with relatively low processing times or structures not suitable to be parallelized. The PC in this research was equipped with 4 physical Intel (R) Core (TM) i7-7700HQ 2.80GHz CPU cores running on Windows 10 operating system.

5.4 Case Study

To evaluate the results of the SCOPF strategy outlined in this Chapter, five medium and large networks were used. These datasets appeared in Challenge 1 of the ARPA GO Competition [157]. Table **5-3** shows the networks used along with the number of buses, generators, loads, lines, reactors, capacitors, and areas in each system. The larger the size of the system, the greater the number of variables in it.

5.5 Results

The results for different network sizes in Table **5-1** are shown in this section. The modifications in the algorithm are compared with the original algorithm from Chapter 4. These comparisons are made in terms of convergence and errors in final cost using a PC-only strategy and another PHC strategies that includes both CPU and CPU architectures for 500 buses network.

5.5.1 Accuracy and Convergence of PHC Architectures

The convergence of the SCOPF implementation is verified using the preliminary results obtained in Chapter 4. Accelerations across PHC architectures using CPU and GPU are compared in terms of number of iterations and final cost for both the base case and contingency case. The results were confirmed by medium-sized network 1. The error during each algorithm iteration was less than $\Delta P < 10^{-10}$, $\Delta Q < 10^{-10}$, $\Delta v_m < 10^{-10}$ and $\Delta c < 10^{-7}$ for the first 5 iterations as shown in Table **5-4**. The above confirms that

Table 5-4: Cost and errors in serial and parallel algorithm execution using network 1

Iteration	Cost Serial	ΔCost	ΔVm	ΔVa	ΔP	ΔQ
1	1,302,553.42	1.35e-07	5.66e-15	1.32e-14	1.26e-10	8.60e-10
2	471,034.72	7.08e-07	5.21e-15	3.10e-14	9.60e-11	9.98e-10
3	471,112.28	5.19e-06	5.51e-15	3.52e-14	9.29e-11	8.79e-10
4	471,034.72	7.08e-07	5.48e-15	3.30e-14	9.91e-11	8.26e-10
5	471,640.67	8.62e-07	5.60e-15	3.31e-14	9.31e-11	6.63e-10

Table 5-5: Average Time for first iteration of the complete SCOPF (s)

Buses	mp Serial	mp Par	GPU Serial	GPU Par
500	535.77	13.91	117.97	64.86
4,918	1,304.06	386.95	2,741.35	715.24
11612	7,713.68	2,405.68	11,531.30	2,310.95
19,402	–	5,131.94	–	4,848.93
30,000	–	17,202.84	–	5,270.00

parallelization using either CPU or GPU will only impact the algorithm's execution time and not the optimal solution.

5.5.2 PHC Architectures Performance

The application of PHC frameworks to SCOPF problems is tested in this section. For this purpose, different computing architectures including CPU and partial GPU are implemented and tested. The networks in Table **5-3** are used to test the performance of the algorithm in Figure **5-1** under different computational architectures. The networks will include the evaluation of all the contingencies set based on system size and contingency screening procedure. The strategy follows recommendations included in [27] to reduce calculation times under limited computational resources. Total required time for the solution of first iteration of each system is shown in Table **5-5**.

Various strategies were implemented following the flow diagram in Figure **5-1**. The OPF solution is carried out in CPU architecture and the contingency evaluation that requires excessive calculation time is carried out using different parallel computing structures. The main differences in the evaluations are: 1) The scenarios are evaluated sequentially or serially 2) the parallelization using CPU cores is based on developed Matlab environments 3) In the algorithms that use partially parallelized structures using GPU, the Client - Server structure described in Section 5.3.5.1 is followed. All strategies use the same algorithm, however, they

change their computational execution structure. Table **5-5** and **5-6** describes the results for the set of strategies evaluated. *mp Serial* uses serial CPU cores, *mp Par* uses Parallel CPU cores, *GPU Serial* uses GPU threads and *GPU Par* uses GPU threads.

5.5.2.1 Regular CPU One Core

For this PHC structure a single CPU core is used so that the algorithm is evaluated serially. Thus the activities are evaluated one after the other. Times for one iteration of medium and large-sized systems are shown in column 2 of Table **5-5**. Networks greater than 20,000 nodes require excessive times even for one iteration. The solution time includes the initial OPF solution, contingency evaluation and constraint handling times. This test is a reference for benchmark in the performance of architectures that include parallel processing. The acceleration achieved for one algorithm iteration in medium and large networks is shown in Table **5-6**. The acceleration achieved follows the relationship in equation 5-2. Where T_x is the time reported in the columns and T_y in the rows.

$$S_{x,y} = T_x/T_y \tag{5-2}$$

Large power network of 19,402 and 30,000 buses are not executed serially since the required time was higher than 4 hours.

5.5.2.2 Parallel CPU

In order to reduce the execution time encountered with the serial strategy, a parallel strategy using multiple CPU cores is employed. The strategy is implemented in Matlab. The entire algorithm that follows this strategy uses CPU cores for both serial and parallelized activities. Each CPU core executes a contingency for simultaneous evaluation. For this test, a total of 4 cores are used, which are those existing in the processing unit. The times found for one SCOPF algorithm operation are shown in column 4 of Table **5-5**. The acceleration achieved is shown in cells formed by combinations of columns and rows in Table **5-6**. In this case, the acceleration of the algorithm executed in parallel with respect to that executed in series is shown in the element in mp Par-mp Serial of each network tested. In this case, accelerations of 38,52 are shown in medium-sized networks and more than 3 in large networks using only CPU cores. The acceleration obtained varies according to the size of the network. This is observed as a result of the solution time of each network, which when distributed is greatly reduced.

5.5.2.3 Serial GPU

Another PHC structure that includes CPU cores and GPU threads is evaluated. This time the vectorized strategy of Chapter 3 and Section 5.3.5.3 are included. The parallelized activities are the ones in green in Figure **5-1**. The initial solution includes contingency

Table 5-6: Speed up under different strategies for first algorithm iteration

Buses	Strategy	mp serial	mp par	GPU Serial	GPU Par
500	**mp serial**	1.00	38.52	4.54	8.26
	mp par	0.03	1.00	0.12	0.21
	GPU Serial	0.22	8.48	1.00	1.82
	GPU Par	0.12	4.66	0.55	1.00
4918	**mp serial**	1.00	3.37	0.48	1.82
	mp par	0.30	1.00	0.14	0.54
	GPU Serial	2.10	7.08	1.00	3.83
	GPU Par	0.55	1.85	0.26	1.00
11612	**mp serial**	1.00	3.21	0.67	3.34
	mp par	0.31	1.00	0.21	1.04
	GPU Serial	1.49	4.79	1.00	4.99
	GPU Par	0.30	0.96	0.20	1.00
19,402	**mp serial**	–	–	–	–
	mp par	–	1	–	1.0584
	GPU Serial	–	–	–	–
	GPU Par	-	0.9449	-	1
30000	**mp serial**	–	–	–	–
	mp par	–	1	–	3.2643
	GPU Serial	–	–	–	–
	GPU Par	–	0.3063	–	1

assessment that creeps into the GPU environment. In the GPU the power flow equations are parallelized. Inside the GPU concurrency is not considered , which does not allow the processing of multiple contingencies inside the GPU. In this case, larger network sizes result in longer computation times. One of the relevant parameters is the transfer times associated with high latency in the CPU-GPU channel. This time has a great impact on the total solution time. In most networks the transfer time degrades the accelerations achieved during other stages inside the GPU. The results are shown in greater times and accelerations less than 1 as shown in Tables **5-5** and **5-6**.

Large power network of 19,402 and 30,000 buses are not executed serially since the required time was higher than 4 hours.

5.5.2.4 Concurrent GPU

In order to run multiple networks, representing various contingencies, the architecture shown in Section 5.3.5.1 is implemented. The architecture allows to use the maximum capacity of the GPU, executing multiple contingencies that come from different kernels of available CPUs. 4 networks are run simultaneously and launched from CPU cores. Solution times are efficient in very large networks such as 20,000 and 30,000 including transfer times as described in Tables **5-5** and **5-6**.

5.5.3 Sparse Linear Solver Performance

Three linear solvers were evaluated for step 9 of Algorithm 2. LU3 from mplinsolve (matpower solver), Intel MKL Pardiso via pypardiso interface [176] and NICSLU [177]. 50 pairs of Jacobian and mismatch vectors were taken from each power system network. On average, mplinsolve took 71.42 ms to solve the system while pypardiso and NICSLU needed 89.48 and 49.12 ms respectively.

5.5.4 Discussion

The results in Tables **5-5** and **5-6** show that PHC structures are adequate with current computational resources to solve SCOPF problems. The results are shown in networks of 500 to 30,000 buses using a considerable amounts of computational resources in demanding processes such as contingency evaluation. PHC structures are found to be suitable for the solution of medium and large power grids. However, the results show that structures with only computer cores are more efficient for medium-sized networks. Meanwhile, structures that use hybrid CPU and GPU structures have better results in large power networks, in this case greater than 20,000 buses, as shown in Table **5-7**.

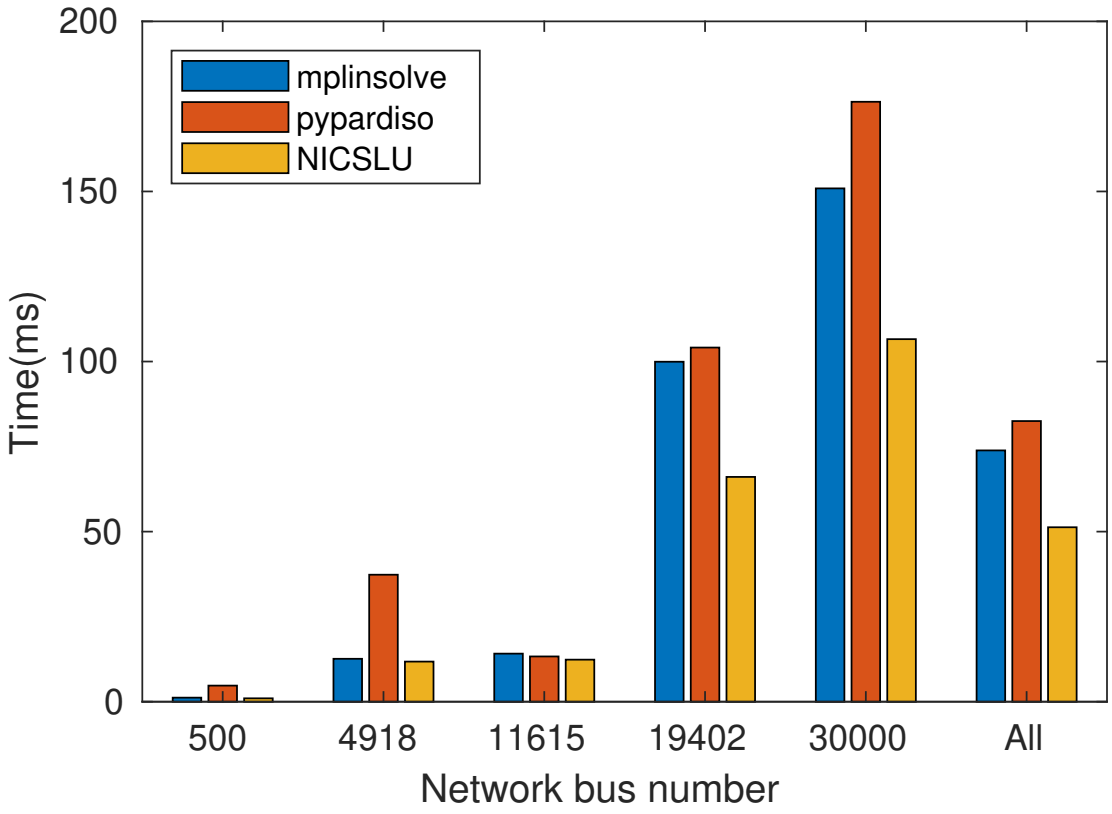

Figure 5-5: Average solver time for different power system sizes

Table 5-7: Results for one iteration of the SCOPF algorithm using parallel architectures for CPU and CPU-GPU

500											
	Initialize	Ybus	Sbus	mis	f	convergence	Jacobian	Solver	updateV	CPU-GPU Transfer	Total
CPU (ms)	0.16	0.89	0.12	0.09	0.10	0.03	1.25	2.63	0.21	0.00	5.48
CPU (%)	2.92	16.31	2.17	1.61	1.90	0.48	22.75	47.98	3.88	0.00	100
GPU (ms)	0.15	51.48	1.93	0.02	0.01	0.02	44.79	2.25	0.25	8.80	109.7
GPU (%)	0.14	46.93	1.76	0.02	0.01	0.02	40.84	2.05	0.23	8.02	100
4,918											
	Initialize	Ybus	Sbus	mis	f	convergence	Jacobian	Solver	updateV	CPU-GPU Transfer	Total
CPU (ms)	0.64	6.65	0.40	0.66	0.51	0.04	10.48	28.14	1.44	0.00	48.96
CPU (%)	1.31	13.58	0.81	1.35	1.03	0.07	21.40	57.49	2.94	0.00	100
GPU (ms)	0.77	56.28	12.91	0.03	0.01	0.04	36.30	26.94	0.25	13.82	147.35
GPU (%)	0.52	38.20	8.76	0.02	0.01	0.02	24.63	18.29	0.17	9.38	100
11,612											
	Initialize	Ybus	Sbus	mis	f	convergence	Jacobian	Solver	updateV	CPU-GPU Transfer	Total
CPU (ms)	1.40	20.09	0.86	1.92	1.37	0.06	34.96	240.21	3.85	0.00	304.72
CPU (%)	0.46	6.59	0.28	0.63	0.45	0.02	11.47	78.83	1.26	0.00	100
GPU (ms)	1.72	39.84	9.07	0.06	0.01	0.07	31.37	184.14	0.30	26.47	293.05
GPU (%)	0.59	13.59	3.10	0.02	0.00	0.02	10.70	62.84	0.10	9.03	100
19,402											
	Initialize	Ybus	Sbus	mis	f	convergence	Jacobian	Solver	updateV	CPU-GPU Transfer	Total
CPU (ms)	2.58	37.45	1.41	3.11	2.31	0.08	71.24	331.38	6.60	0.00	456.16
CPU (%)	0.56	8.21	0.31	0.68	0.51	0.02	15.62	72.65	1.45	0.00	100
GPU (ms)	3.20	36.36	8.26	0.06	0.02	0.10	32.39	281.34	0.36	47.56	409.65
GPU (%)	0.78	8.88	2.02	0.02	0.00	0.02	7.91	68.68	0.09	11.61	100
30,000											
	Initialize	Ybus	Sbus	mis	f	convergence	Jacobian	Solver	updateV	CPU-GPU Transfer	Total
CPU (ms)	3.70	43.23	2.12	18.32	15.14	0.43	609.62	5343.02	62.04	0.00	6,097.62
CPU (%)	0.06	0.71	0.03	0.30	0.25	0.01	10.00	87.62	1.02	0.00	100
GPU (ms)	4.64	35.72	7.87	0.43	0.10	0.59	201.66	1601.18	2.73	121.35	1,976.27
GPU (%)	0.23	1.81	0.40	0.02	0.01	0.03	10.20	81.02	0.14	6.14	100

In the case of medium-sized networks, it is observed that the activities that are executed more efficiently in CPU than in GPU are the calculations Y_{bus} and the Jacobian matrix. This way they run faster on CPU. The other activities show lower GPU times. At this network size, a high negative impact on information transfer times between CPU-GPUs is observed. This is the result of the low time required for other activities and the high latency in the transfer process. The results also show a high impact on the solution time of the systems of equations using different solvers. Using the specified algorithm the more efficient solver was NICSLU followed by MATPOWER and PyPardiso.

Results on large networks show faster responses for the execution of PHC structures that use CPU and GPU. Most activities on large networks are performed more efficiently on GPUs. Although the transfer time has an impact on the total execution time, the percentage that this requires with respect to the total is low. The result is a PHC structure optimized for GPU execution using constraint handling. The results for the solution of the equations generated by the formulation also show a better performance of NICSLU over the MATPOWER and PyPardiso solvers.

5.6 Conclusions

This Chapter proposes the acceleration of the algorithm shown in Chapter 4 in order to accelerate the solution of a SCOPF problem. The strategy shown includes the use of an PHC architecture capable of accelerating the solution time of the proposed strategy. The algorithm and the proposed architecture are tested using large networks that include a considerable number of buses, loads, lines and contingencies. The systems used exceed the size of the common systems in the literature. The PHC framework uses a hybrid of CPU and GPU architectures to speed up the process of calculating power flows in the contingency assessment stage.

The algorithm is based on a bi-level algorithm that solves a base problem that includes the normal operation of the system and a second stage of the system under contingency. Once the base case is solved, the limits of each of the existing constraints are updated not only to guarantee an optimal operation in a steady state but also once each of the different contingencies have been applied. Each contingency updates the limits of the different constraints. Since the algorithm is mainly based on power flows, it is suitable for implementation using the CPU and GPU framework from Chapter 3.

The architecture is implemented in a laptop, however, its structure is suitable to be applied in clusters of CPUs and GPUs. The structure is evaluated with Matlab and CUDA in the case of the stages that include the use of GPUs.

6 Contributions and Concluding Remarks

6.1 Contributions

In this thesis, a presentation of the High Performance Computing (HPC) architectures applied to the analysis of power systems, particularly Security Constrained Optimal Power Flow (SCOPF) is made. The research performs a review of the applications of HPC structures in power systems analysis, showing a limited application of these structures to SCOPF problems in Chapter 2.

The first contribution of this research is a proposed vectorized power flow architecture for local microgrid control. Chapter 3 shows the implementation of a vectorized algorithm to solve large power networks using low cost embedded computers using an HC architecture. The proposal makes a feasible implementation for a local controller to guarantee secure operation of isolated or grid-connected microgrids. The strategy works in near real time.

As a second contribution of this work, an efficient algorithm for the solution of large and complex SCOPF problems is implemented using secure constraint handling using a heuristic algorithm. Chapter 4 shows successful results for real time operation in large power systems. The architecture was based on PC architectures that involve computer clustering techniques. The strategy was tested on near real time with appropriate outcomes.

As a third contribution of this thesis, a HPC that includes CPU cores and GPU threads is implemented. Chapter 5 shows the results exploit the potential of CPU cores and vectorized GPU structures for contingency evaluation. The responses show that hybrid architectures are cost effective solutions to accelerate the solution of SCOPF problems. The results show that CPU and GPU are feasible and appropriate architecture for the solution of SCOPF in real-time.

6.2 Answering the Research Questions

What specific SCOPF features and formulations are suitable for being parallelized using PHC architectures?

The research shows different approaches for the solution of the SCOPF problem. The results shows that PHC are effective strategies for the solution of the problem. Since the security feature is one of the main features of the problem, the evaluation of multiple contingency in large power systems is a burdensome when system of thousands of buses and lines are analyzed. The research and results showed that contingency evaluation as the solution of large and complex problem formulation are the main features to be accelerated in SCOPF using PHC architectures.

How much faster can the evaluation be completed using PHC integration to solve SCOPF problems in comparison with a traditional solution using only CPU cores?

Multiple strategies showed the application of parallel structures for the solution of different power system networks. Previous researcher results show promising applications of CPU cores and GPU structures to solve power system problems; however, limited application had been shown in SCOPF . The application show that GPU implementation allow faster result than single CPU cores in large power networks. This condition makes the application of PHC a cost-effective solution to accelerate the algorithm responses. The application show responses x3.26 faster using a conventional PC. The strategy is scalable to CPU and GPU clusters.

Does the strategy satisfy the on-line security times and optimum solutions required by system operators?

The responses of the SCOPF using PHC structures show feasible solutions in terms of time and optimal final points. The implementation in this research avoids multiple OPFs and uses power flow runs to perform constraint handling. The results show a considerable reduction in algorithm execution time making the strategy adequate for real-time operation. The final optimal solution shows successful results in large power networks, making the strategy feasible in terms of optimal point and final computing time. The strategy is scalable in industrial environments.

6.3 Directions for Future Research

Some ideas for future research are listed below:

- In the development of this thesis, a PC proposal is made using CPU cores to solve non-convex problems that include LEMs. Although the strategy was appropriate, parallel clusters could be implemented on GPUs in order to increase performance and further reduce processing time.

- Current work shows promising results including a local GPU and multiple cores on a personal computer and a Workstation. The results can be exploited and reproduced in clusters of CPUs and GPUs. The solution can also be extended to the use of Cloud or Fog services and integration of CPUs and GPUs for the SCOPF solution, which would accelerate the calculation and solution process of the system in a more considerable way.

- During this work, processes that consume a large number of resources and time during the solution of SCOPF have been accelerated. These activities included the execution of contingencies, which in large networks are exhaustive due to the number of lines, transformers, generators, loads and reactive compensations. During the development of the thesis, different tests were carried out in order to find suitable solvers of equations for the solution of OPF using vectorized structures. However, the results were not competitive in terms of execution time. Looking for more efficient structures in the solver could significantly reduce execution times.

References

[1] Q. Wang, *Risk-based Security-Constrained Optimal Power Flow: Mathematical Fundamentals, Computational Strategies, Validation, and use within Electricity Markets*. PhD thesis, Iowa State University, 2013.

[2] D. Page, *A Practical Introduction to Computer Architecture*. Springer, 2009.

[3] NVIDIA, "NVIDIA Turing GPU," *White Paper*, 2018.

[4] S. Rennich, "Cuda c/c++ streams and concurrency," tech. rep., 2012.

[5] G. Wood, A; Wollenberg, B; Sheblé, *Power Generation, Operation and Control*. Wiley, 2014.

[6] F. Garcia, N. D. 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," *IEEE Power and Energy Society General Meeting*, pp. 1–8, 2012.

[7] Z. Li and F. Yang, *Advanced metering infrastructure and graphics processing unit technologies in electric distribution networks*. No. 9789811070006, Springer Singapore, 2018.

[8] NERC, "Standard TPL-001-4 — Transmission System Planning Performance Requirements," vol. 2, 2014.

[9] J. K. Debnath, W. K. Fung, A. M. Gole, and S. Filizadeh, "Simulation of large-scale electrical power networks on graphics processing units," *2011 IEEE Electrical Power and Energy Conference, EPEC 2011*, pp. 199–204, 2011.

[10] J. Baranowski and D. J. French, "Operational use of contingency analysis at PJM," *IEEE Power and Energy Society General Meeting*, pp. 13–16, 2012.

[11] TOP500, "Home — TOP500 Supercomputer Sites," 2020.

[12] R. Gnanavignesh and U. J. Shenoy, "Parallel Sparse LU Factorization of Power Flow Jacobian using GPU," *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, vol. 2019-Octob, no. C, pp. 1857–1862, 2019.

[13] I. Araújo, V. Tadaiesky, D. Cardoso, Y. Fukuyama, and Á. Santana, "Simultaneous parallel power flow calculations using hybrid CPU-GPU approach," *International Journal of Electrical Power and Energy Systems*, vol. 105, no. February 2018, pp. 229–236, 2019.

[14] Q. Shi, C. Yuan, W. Feng, G. Liu, R. Dai, Z. Wang, and F. Li, "Enabling Model-Based LTI for Large-Scale Power System Security Monitoring and Enhancement with Graph-Computing-Based Power Flow Calculation," *IEEE Access*, vol. 7, pp. 167010–167018, 2019.

[15] P. Duan, S. Xu, H. Chen, X. Yang, S. Wang, and E. Hu, "High Performance Computing (HPC)for Advanced Power System Studies," *2nd IEEE Conference on Energy Internet and Energy System Integration, EI2 2018 - Proceedings*, pp. 1–9, 2018.

[16] K. Tang, S. Dong, B. Zhu, Q. Ni, and Y. Song, "GPU-Based Real-time N-1 AC Power Flow Algorithm with Preconditioned Iterative Method," *IEEE Power and Energy Society General Meeting*, vol. 2018-Augus, pp. 1–5, 2018.

[17] G. Ruetsch and B. Oster, "Getting Started with CUDA What is CUDA ?," *Materials*, vol. 17, no. 4, pp. 223–224, 2008.

[18] T. Soyata, *GPU Parallel Program Development Using CUDA*. 2018.

[19] L. Platbrood, H. Crisciu, F. Capitanescu, and L. Wehenkel, "Solving very large-scale security-constrained optimal power flow problems by combining iterative contingency selection and network compression," *17th Power Systems Computation Conference, PSCC 2011*, 2011.

[20] F. Capitanescu, M. Glavic, D. Ernst, and L. Wehenkel, "Applications of security-constrained optimal power flows," *Modern Electric Power Systems Symposium, MEPS06*, p. 7, 2006.

[21] F. Capitanescu, "Critical review of recent advances and further developments needed in AC optimal power flow," *Electric Power Systems Research*, vol. 136, pp. 57–68, 2016.

[22] O. Alsaç, J. Bright, M. Prais, and B. Stott, "Further developments in lp-based optimal power flow," *IEEE Transactions on Power Systems*, vol. 5, no. 3, pp. 697–711, 1990.

[23] D. Rodriguez-Medina, D. Gomez, S. Rivera, and J. Gers, "A fast decomposition method to solve a security-constrained optimal power flow (scopf) empowered by heterogeneous and parallel computing (hpc) (under review)," *PES General Meeting*, pp. 52812–52824, 2022.

[24] D. Rodriguez, D. Alvarez, D. Gomez, J. Gers, and S. Rivera, "Low-cost analysis of load flow computing using embedded computer empowered by gpu," *Proceedings - IEEE PES ISGT NA 2021: "Technology Solutions for an Evolving Grid"*, 02 2021.

[25] D. Rodriguez, D. Gomez, D. Alvarez, and S. Rivera, "A review of parallel heterogeneous computing algorithms in power systems," *Algorithms*, vol. 14, no. 10, 2021.

[26] D. Rodriguez, A. Angulo, D. F. Gomez, A. David, J. Gil, and S. Rivera, "Smart Microgrids Operation Considering Expert Knowledge and Ensembled Based Metaheuristic Optimization Algorithms *(Under Review)*," *International Journal of Electrical and Computer Science*, vol. 12, 2021.

[27] T. Valencia-Zuluaga, D. Agudelo-Martinez, D. Arango-Angarita, C. Acosta-Urrego, S. Rivera, D. Rodriguez-Medina, and J. Gers, "A Fast Decomposition Method to Solve a Security-Constrained Optimal Power Flow (SCOPF) Problem through Constraint Handling," *IEEE Access*, vol. 9, pp. 52812–52824, 2021.

[28] A. Angulo, D. Rodríguez, W. Garzón, D. F. Gómez, A. Al Sumaiti, and S. Rivera, "Algorithms for bidding strategies in local energy markets: Exhaustive search through parallel computing and metaheuristic optimization," *Algorithms*, vol. 14, no. 9, 2021.

[29] J. Ramírez-Romero, D. Medina, and S. Rivera, "Teaching using a synchronous machine virtual laboratory," *Global Journal of Engineering Education*, vol. 22, pp. 123–130, 06 2020.

[30] J. Garcia-Guarin, D. Rodriguez, D. Alvarez, S. Rivera, C. Cortes, A. Guzman, A. Bretas, J. R. Aguero, and N. Bretas, "Smart microgrids operation considering a variable neighborhood search: The differential evolutionary particle swarm optimization algorithm," *Energies*, vol. 12, no. 16, pp. 1–13, 2019.

[31] S. Vargas, D. Rodriguez, and S. Rivera, "Mathematical Formulation and Numerical Validation of Uncertainty Costs for Controllable Loads," *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería*, vol. 12, 2019.

[32] S. Rivera, D. Rodriguez, and I. Erlich, "2018 Grid Optimization Competition Evaluating the Performance of Modern Heuristic Optimizers on Stochastic Optimization Problems applied to Smart Grids Test bed A : Stochastic OPF in Presence of Renewable Energy and Controllable Loads," *Intelligent Systems Subcommittee Power System Analysis, Computing, and Economic Committee*, no. January, 2018.

[33] J. Arevalo, D. Medina, J. Rueda, and S. Rivera, "2018 competition on operational planning of sustainable power systems: Testsbeds and results," *WSEAS Transactions on Power Systems*, vol. 14, pp. 98–106, 08 2019.

[34] D. Rodriguez, D. Gomez, W. Garzon, D. Alvarez, S. Rivera, and J. Gers, "Posicionamiento Óptimo de cuadrillas basado en estadísticas de Tránsito de Google Maps e Indicadores de Confiabilidad," 2018.

[35] D. Rodriguez, J. M. Gers, T. Valencia, C. Acosta, D. Agudelo, and D. Arango, "Ensembled Method: Constraints Relaxation With Analytical Optimization With Combined Heuristic Method," tech. rep., 2020.

[36] D. Rodriguez and T. Valencia, "Posicionamiento Óptimo de cuadrillas basado en estadísticas de Tránsito de Google Maps e Indicadores de Confiabilidad," 2018.

[37] J. García, D. Rodríguez, and S. Rivera, "Herramienta para la Programación de Redes Inteligentes con Recursos Energéticos de Alta Incertidumbre," 2019.

[38] NERC, *Reliability Assessment Guidebook*. 1 ed., 2010.

[39] N. Garcia, "Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach," *IEEE PES General Meeting, PES 2010*, no. 5, pp. 1–4, 2010.

[40] ARPA, "About the Competition — Grid Optimization Competition," 2019.

[41] S. Huang and V. Dinavahi, "Fast Batched Solution for Real-Time Optimal Power Flow with Penetration of Renewable Energy," *IEEE Access*, vol. 6, pp. 13898–13910, 2018.

[42] V. H. Hinojosa and F. Gonzalez-Longatt, "Preventive security-constrained DCOPF formulation using power transmission distribution factors and line outage distribution factors," *Energies*, vol. 11, no. 6, pp. 1–13, 2018.

[43] Y. Yu and P. Luh, "Scalable corrective security-constrained economic dispatch considering conflicting contingencies," *International Journal of Electrical Power and Energy Systems*, vol. 98, no. December 2017, pp. 269–278, 2018.

[44] Y. Yang and Y. Feng, "Large-scale preventive security constrained optimal power flow based on compensation method," *IEEE Power and Energy Society General Meeting*, vol. 2015-Septe, 2015.

[45] F. Capitanescu, J. L. Martinez Ramos, P. Panciatici, D. Kirschen, A. Marano Marcolini, L. Platbrood, and L. Wehenkel, "State-of-the-art, challenges, and future trends in security constrained optimal power flow," *Electric Power Systems Research*, vol. 81, no. 8, pp. 1731–1741, 2011.

[46] H. Harsan, N. Hadjsaid, and P. Pruvot, "Cyclic Security Analysis for Security Constrained Optimal Power Flow," *IEEE Transactions on Power Systems*, vol. 12, no. 2, pp. 948–953, 1997.

[47] Q. Wang, J. D. McCalley, T. Zheng, and E. Litvinov, "Solving corrective risk-based security-constrained optimal power flow with Lagrangian relaxation and Benders decomposition," *International Journal of Electrical Power and Energy Systems*, vol. 75, pp. 255–264, 2016.

[48] Y. Li and J. D. McCalley, "Decomposed SCOPF for improving efficiency," *IEEE Transactions on Power Systems*, vol. 24, no. 1, pp. 494–495, 2009.

[49] Q. Wang, J. D. McCalley, T. Zheng, and E. Litvinov, "A computational strategy to solve preventive risk-based security-constrained OPF," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1666–1675, 2013.

[50] V. H. Hinojosa, "Comparative Corrective and Preventive Security-Constrained DCOPF Problems Using Linear Shift-Factors," *Energies*, pp. 1–16, 2020.

[51] J. Mohammadi, G. Hug, and S. Kar, "A benders decomposition approach to corrective security constrained OPF with power flow control devices," *IEEE Power and Energy Society General Meeting*, 2013.

[52] S. Huang and V. Dinavahi, "Performance analysis of GPU-accelerated fast decoupled power flow using direct linear solver," *2017 IEEE Electrical Power and Energy Conference, EPEC 2017*, vol. 2017-Octob, no. 1, pp. 1–6, 2017.

[53] M. Wang, Y. Chen, and S. Huang, "GPU-based Power Flow Analysis with Continuous Newton ' s Method," *IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pp. 1–5, 2017.

[54] L. Y. Kyaw and S. Phyu, "Scheduling Methods in HPC System: Review," *2020 IEEE Conference on Computer Applications, ICCA 2020*, pp. 1–6, 2020.

[55] L. R, P. M, and B. C, *Computational Physics*. 2007.

[56] P. Marksteiner, "High-performance computing - An overview," *Computer Physics Communications*, vol. 97, no. 1-2, pp. 16–35, 1996.

[57] H. Andrade and I. Crnkovic, "A Review on Software Architectures for Heterogeneous Platforms," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 2018-December, pp. 209–218, 2018.

[58] P. N. Glaskowsky, "NVIDIA ' s Fermi : The First Complete GPU Computing Architecture," no. September, 2009.

[59] M. Marin, G.-e. P. Flow, and A. Distributed, *GPU-Enhanced power flow analysis*. PhD thesis, UNIVERSITE DE PERPIGNAN VIA DOMITIA-UNIVERSITY COLLEGE DUBLIN, 2016.

[60] Z. Feng, X. Zhao, and Z. Zeng, "Robust parallel preconditioned power grid simulation on gpu with adaptive runtime performance modeling and optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 562–573, 2011.

[61] Z. Li, J. Zhu, and F. Yang, "How far is the GPU technology from practical power system applications?," *IEEE Power and Energy Society General Meeting*, vol. 2014-Octob, no. October, 2014.

[62] X. Li, F. Li, and S. Member, "GPU-based Fast Decoupled Power Flow with Preconditioned Iterative Solver and Inexact Newton Method," *IEEE Power & Energy Society General Meeting*, vol. 8950, no. c, pp. 1–1, 2017.

[63] V. Roberge, M. Tarbouchi, and F. Okou, "Parallel power flow on graphics processing units for concurrent evaluation of many networks," *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 1639–1648, 2017.

[64] V. Jalili-Marandi and V. Dinavahi, "Simd-based large-scale transient stability simulation on the graphics processing unit," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1589–1599, 2010.

[65] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 7, pp. 1255–1266, 2012.

[66] Z. Zhou and V. Dinavahi, "Parallel massive-thread electromagnetic transient simulation on gpu," *IEEE Transactions on Power Delivery*, vol. 29, no. 3, pp. 1045–1053, 2014.

[67] Y. Song, Y. Chen, S. Huang, Y. Xu, Z. Yu, and J. R. Marti, "Fully gpu-based electromagnetic transient simulation considering large-scale control systems for system-level studies," *IET Generation, Transmission Distribution*, vol. 11, no. 11, pp. 2840–2851, 2017.

[68] N. Lukač and B. Žalik, "Gpu-based roofs' solar potential estimation using lidar data," *Computers Geosciences*, vol. 52, pp. 34 – 41, 2013.

[69] R. C. Green, L. Wang, and M. Alam, "Applications and trends of high performance computing for electric power systems: Focusing on smart grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 922–931, 2013.

[70] G. Capizzi, G. Lo Sciuto, C. Napoli, and E. Tramontana, "Advanced and adaptive dispatch for smart grids by means of predictive models," *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 6684–6691, 2018.

[71] G. Zhou, X. Zhang, Y. Lang, R. Bo, Y. Jia, J. Lin, and Y. Feng, "A novel GPU-accelerated strategy for contingency screening of static security analysis," *International Journal of Electrical Power and Energy Systems*, vol. 83, pp. 33–39, 2016.

[72] H. Karimipour and V. Dinavahi, "Extended kalman filter-based parallel dynamic state estimation," *IEEE Transactions on Smart Grid*, vol. 6, no. 3, pp. 1539–1549, 2015.

[73] W. Qiu, Q. Tang, J. Liu, Z. Teng, and W. Yao, "Power quality disturbances recognition using modified s transform and parallel stack sparse auto-encoder," *Electric Power Systems Research*, vol. 174, p. 105876, 2019.

[74] Z. Liu, X. Li, L. Wu, S. Zhou, and K. Liu, "Gpu-accelerated parallel coevolutionary algorithm for parameters identification and temperature monitoring in permanent magnet synchronous machines," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 5, pp. 1220–1230, 2015.

[75] V. Schwarzer and R. Ghorbani, "New opportunities for large-scale design optimization of electric vehicles using gpu technology," in *2011 IEEE Vehicle Power and Propulsion Conference*, pp. 1–6, 2011.

[76] G. Zhou, R. Bo, L. Chien, X. Zhang, S. Yang, and D. Su, "Gpu-accelerated algorithm for online probabilistic power flow," *IEEE Transactions on Power Systems*, vol. 33, pp. 1132–1135, Jan 2018.

[77] Z. Chen, L. Shen, Y. Zhao, and C. Yang, "Parallel algorithm for real-time contouring from grid dem on modern gpus," *Science China Technological Sciences*, vol. 53, pp. 33–37, May 2010.

[78] T. He, K. Meng, Z.-Y. Dong, Y.-T. Oh, and Y. Xu, "Use of high-performance graphics processing units for power system demand forecasting," *Journal of Electrical Engineering and Technology*, vol. 5, p. 363–370, Jan 2010.

[79] F. Milano, "Small-signal stability analysis of large power systems with inclusion of multiple delays," *IEEE Transactions on Power Systems*, vol. 31, no. 4, pp. 3257–3266, 2016.

[80] B. Shang, Y. Xu, C. Zhang, Y. Chen, Z. Liu, L. Lin, C. Xu, and J. Yu, "Gpu-accelerated batch solution for short-circuit current calculation of large-scale power systems," in *2019 IEEE 3rd International Electrical and Energy Conference (CIEEC)*, pp. 1743–1748, 2019.

[81] J. S. Chai, N. Zhu, A. Bose, and D. J. Tylavsky, "Parallel newton type methods for power system stability analysis using local and shared memory multiprocessors," *IEEE Transactions on Power Systems*, vol. 6, no. 4, pp. 1539–1545, 1991.

[82] J. Shu, Wei Xue, and Weimin Zheng, "A parallel transient stability simulation for power systems," *IEEE Transactions on Power Systems*, vol. 20, no. 4, pp. 1709–1717, 2005.

[83] P. Aristidou, D. Fabozzi, and T. Van Cutsem, "Dynamic simulation of large-scale power systems using a parallel schur-complement-based decomposition method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2561–2570, 2014.

[84] S. K. Khaitan, J. D. McCalley, and A. Somani, "Proactive task scheduling and stealing in master-slave based load balancing for parallel contingency analysis," *Electric Power Systems Research*, vol. 103, pp. 9 – 15, 2013.

[85] S. K. Khaitan and J. D. McCalley, "Scale: A hybrid mpi and multithreading based work stealing approach for massive contingency analysis in power systems," *Electric Power Systems Research*, vol. 114, pp. 118 – 125, 2014.

[86] Jun Qiang Wu and A. Bose, "Parallel solution of large sparse matrix equations and parallel power flow," *IEEE Transactions on Power Systems*, vol. 10, no. 3, pp. 1343–1349, 1995.

[87] X. Wang, S. G. Ziavras, C. Nwankpa, J. Johnson, and P. Nagvajara, "Parallel solution of newton's power flow equations on configurable chips," *International Journal of Electrical Power Energy Systems*, vol. 29, no. 5, pp. 422 – 431, 2007.

[88] J. Baek, Q. H. Vu, J. K. Liu, X. Huang, and Y. Xiang, "A secure cloud computing based framework for big data information management of smart grid," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 233–244, 2015.

[89] J. Soares, M. A. F. Ghazvini], Z. Vale, and P. [de Moura Oliveira], "A multi-objective model for the day-ahead energy resource scheduling of a smart grid with high penetration of sensitive loads," *Applied Energy*, vol. 162, pp. 1074 – 1088, 2016.

[90] G. N. Korres, A. Tzavellas, and E. Galinas, "A distributed implementation of multi-area power system state estimation on a cluster of computers," *Electric Power Systems Research*, vol. 102, pp. 20 – 32, 2013.

[91] Y. Fukuyama and Hsaio-Dong Chiang, "A parallel genetic algorithm for generation expansion planning," *IEEE Transactions on Power Systems*, vol. 11, no. 2, pp. 955–961, 1996.

[92] A. Rami, A. Zeblah, H. Hamdaoui, Y. Massim, and F. Harrou, "An efficient artificial immune algorithm for power system reliability optimisation," *International Journal of Power and Energy Conversion*, vol. 1, no. 2-3, pp. 178–197, 2009. cited By 7.

[93] C. Dufour, V. Jalili-Marandi, and J. Bélanger, "Real-time simulation using transient stability, electromagnetic transient and fpga-based high-resolution solvers," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 283–288, 2012.

[94] J. Ma, K. L. Man, S.-U. Guan, T. O. Ting, and P. W. H. Wong, "Parameter estimation of photovoltaic model via parallel particle swarm optimization algorithm," *International Journal of Energy Research*, vol. 40, no. 3, pp. 343–352, 2016.

[95] F. Sato, A. Garcia, A. Monticelli, and A. B. Alves], "Distributed short-circuit analysis in heterogeneous computer networks," *International Journal of Electrical Power Energy Systems*, vol. 22, no. 2, pp. 129 – 136, 2000.

[96] A. K. Zadeh, K. M. Nor, and H. Zeynal, "Multi-thread security constraint economic dispatch with exact loss formulation," in *2010 IEEE International Conference on Power and Energy*, pp. 864–869, 2010.

[97] T. Cui and F. Franchetti, "A multi-core high performance computing framework for probabilistic solutions of distribution systems," in *2012 IEEE Power and Energy Society General Meeting*, pp. 1–6, 2012.

[98] J. Zhang, S. Lin, H. Liu, Y. Chen, M. Zhu, and Y. Xu, "A small-population based parallel differential evolution algorithm for short-term hydrothermal scheduling problem considering power flow constraints," *Energy*, vol. 123, pp. 538 – 554, 2017.

[99] V. Roberge, M. Tarbouchi, and F. Okou, "Optimal power flow based on parallel metaheuristics for graphics processing units," *Electric Power Systems Research*, vol. 140, pp. 344–353, 2016.

[100] G. Geng, Q. Jiang, and Y. Sun, "Parallel transient stability-constrained optimal power flow using gpu as coprocessor," *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1436–1445, 2017.

[101] B. Kim, "A fast distributed implementation of optimal power flow," *IEEE Transactions on Power Systems*, vol. 14, no. 3, pp. 858–864, 1999. cited By 169.

[102] H. R. Cai, C. Y. Chung, and K. P. Wong, "Application of differential evolution algorithm for transient stability constrained optimal power flow," *IEEE Transactions on Power Systems*, vol. 23, no. 2, pp. 719–728, 2008.

[103] M. Abedini, "A novel algorithm for load flow analysis in island microgrids using an improved evolutionary algorithm," *International Transactions on Electrical Energy Systems*, vol. 26, pp. 2727–2743, dec 2016.

[104] Taufik, M. A. Guevara, A. Shaban, and A. Nafisi, "Modeling and Load Flow Analysis of a Microgrid Laboratory.," *International Journal of Smart Grid and Sustainable Energy Technologies*, vol. 3, pp. 103–111, dec 2019.

[105] J. I. Giraldez Miner, F. Flores-Espino, S. MacAlpine, and P. Asmus, "Phase i microgrid cost study: Data collection and analysis of microgrid costs in the united states," *National Renewable Energy Laboratory*, 10 2018.

[106] V. Kloh, D. Yokoyama, A. Yokoyama, G. Silva, M. Ferro, and B. Schulze, "Performance and Energy Efficiency Evaluation for HPC Applications in Heterogeneous Architectures," in *2018 Symposium on High Performance Computing Systems (WSCAD)*, pp. 162–169, IEEE, oct 2018.

[107] S. K. Khaitan, "A survey of high-performance computing approaches in power systems," in *2016 IEEE Power and Energy Society General Meeting (PESGM)*, pp. 1–5, IEEE, jul 2016.

[108] L. Rakai and W. Rosehart, "GPU-accelerated solutions to optimal power flow problems," *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 2511–2516, 2014.

[109] D. Greenwood, K. Lim, C. Patsios, P. Lyons, Y. Lim, and P. Taylor, "Frequency response services designed for energy storage," *Applied Energy*, vol. 203, pp. 115 – 127, 2017.

[110] D. Carreira, G. D. Marques, and D. M. Sousa, "Hybrid energy storage system with a low cost digital control," in *2015 9th International Conference on Compatibility and Power Electronics (CPE)*, pp. 185–190, 2015.

[111] D. Wu, T. Dragicevic, J. C. Vasquez, J. M. Guerrero, and Y. Guan, "Secondary coordinated control of islanded microgrids based on consensus algorithms," in *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 4290–4297, 2014.

[112] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra, "Graphics processing unit (GPU) programming strategies and trends in GPU computing," *Journal of Parallel and Distributed Computing*, vol. 73, pp. 4–13, jan 2013.

[113] H. H. Holm, A. R. Brodtkorb, and M. L. Sætra, "GPU Computing with Python: Performance, Energy Efficiency and Usability," *Computation*, vol. 8, p. 4, jan 2020.

[114] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-Based Python JIT Compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[115] P. Virtanen, R. Gommers, and et al., "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, mar 2020.

[116] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, pp. 12–19, feb 2011.

[117] Q. Wang, J. D. McCalley, T. Zheng, and E. Litvinov, "A computational strategy to solve preventive risk-based security-constrained OPF," *IEEE Transactions on Power Systems*, vol. 28, pp. 1666–1675, may 2013.

[118] S. A. Sadat, D. Haralson, and M. Sahraei-Ardakani, "Security versus computation time in IV-ACOPF with SOCP initialization," in *2018 IEEE International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*, IEEE, jun 2018.

[119] F. Capitanescu, J. M. Ramos, P. Panciatici, D. Kirschen, A. M. Marcolini, L. Platbrood, and L. Wehenkel, "State-of-the-art, challenges, and future trends in security constrained optimal power flow," *Electric Power Systems Research*, vol. 81, pp. 1731–1741, Aug 2011.

[120] F. Capitanescu, "Critical review of recent advances and further developments needed in AC optimal power flow," *Electric Power Systems Research*, vol. 136, pp. 57–68, jul 2016.

[121] D. Phan and J. Kalagnanam, "Some efficient optimization methods for solving the security-constrained optimal power flow problem," *IEEE Transactions on Power Systems*, vol. 29, pp. 863–872, March 2014.

[122] J. Mohammadi, G. Hug, and S. Kar, "A benders decomposition approach to corrective security constrained OPF with power flow control devices," in *2013 IEEE Power & Energy Society General Meeting*, IEEE, 2013.

[123] D. T. Phan and X. A. Sun, "Minimal impact corrective actions in security-constrained optimal power flow via sparsity regularization," *IEEE Transactions on Power Systems*, vol. 30, pp. 1947–1956, jul 2015.

[124] M. Al-Saffar and P. Musilek, "Distributed optimal power flow for electric power systems with high penetration of distributed energy resources," in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, IEEE, may 2019.

[125] R. Louca and E. Bitar, "Robust AC optimal power flow," *IEEE Transactions on Power Systems*, vol. 34, pp. 1669–1681, may 2019.

[126] Y. Li and J. McCalley, "Decomposed SCOPF for improving efficiency," *IEEE Transactions on Power Systems*, vol. 24, pp. 494–495, feb 2009.

[127] F. Capitanescu, M. Glavic, D. Ernst, and L. Wehenkel, "Applications of security-constrained optimal power flows," in *In Proceedings of Modern Electric Power Systems Symposium, MEPS06*, 2006.

[128] S. Sojoudi and J. Lavaei, "Physics of power networks makes hard optimization problems easy to solve," in *2012 IEEE Power and Energy Society General Meeting*, IEEE, jul 2012.

[129] V. Hinojosa and F. Gonzalez-Longatt, "Preventive security-constrained DCOPF formulation using power transmission distribution factors and line outage distribution factors," *Energies*, vol. 11, p. 1497, jun 2018.

[130] Y. Xu, H. Yang, R. Zhang, Z. Dong, M. Lai, and K. Wong, "A contingency partitioning approach for preventive-corrective security-constrained optimal power flow computation," *Electric Power Systems Research*, vol. 132, pp. 132–140, 2016.

[131] Y. Xu, Z. Y. Dong, R. Zhang, K. P. Wong, and M. Lai, "Closure to discussion on "solving preventive-corrective scopf by a hybrid computational strategy"," *IEEE Transactions on Power Systems*, vol. 29, p. 3124–3125, Nov 2014.

[132] M. Javadi, A. E. Nezhad, M. Gough, M. Lotfi, and J. P. Catalao, "Implementation of consensus-ADMM approach for fast DC-OPF studies," in *2019 International Conference on Smart Energy Systems and Technologies (SEST)*, IEEE, sep 2019.

[133] A. Attarha and N. Amjady, "Solution of security constrained optimal power flow for large-scale power systems by convex transformation techniques and taylor series," *IET Generation, Transmission & Distribution*, vol. 10, p. 889–896, Mar 2016.

[134] A. Werner, K. Duwadi, N. Stegmeier, T. M. Hansen, and J.-H. Kimn, "Parallel implementation of ac optimal power flow and time constrained optimal power flow using high performance computing," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, Jan 2019.

[135] E. Karangelos and L. Wehenkel, "An iterative AC-SCOPF approach managing the contingency and corrective control failure uncertainties with a probabilistic guarantee," *IEEE Transactions on Power Systems*, vol. 34, pp. 3780–3790, sep 2019.

[136] S. Lee, W. Kim, and B. H. Kim, "Performance comparison of optimal power flow algorithms for lmp calculations of the full scale korean power system," *Journal of Electrical Engineering and Technology*, vol. 10, p. 109–117, Jan 2015.

[137] Y. Chen, Z. Zhang, Y. Lang, J. Ma, and S. Zheng, "Generalised-fast decoupled state estimator," *IET Generation, Transmission Distribution*, vol. 12, pp. 5928–5938, may 2018.

[138] J. Guo, G. Hug, and O. K. Tonguz, "A case for nonconvex distributed optimization in large-scale power systems," *IEEE Transactions on Power Systems*, vol. 32, pp. 3842–3851, sep 2017.

[139] M. Granada Echeverri, M. J. Rider Flores, and J. R. S. Mantovani, "Dos técnicas de descomposición aplicadas al problema de flujo de potencia óptimo multi-areas," *DYNA*, vol. 77, pp. 303 – 312, 06 2010.

[140] J. Guo, G. Hug, and O. Tonguz, "Asynchronous admm for distributed non-convex optimization in power systems," *arXiv preprint arXiv:1710.08938*, 2017.

[141] M. Bazrafshan, K. Baker, and J. Mohammadi, "Computationally efficient solutions for large-scale security-constrained optimal power flow," 2020.

[142] S. Stankovic and L. Soder, "Optimal power flow based on genetic algorithms and clustering techniques," in *2018 Power Systems Computation Conference (PSCC)*, IEEE, Jun 2018.

[143] A. Zamzam and K. Baker, "Learning optimal solutions for extremely fast ac optimal power flow," *arXiv preprint arXiv:1910.01213*, 2019.

[144] I. Ghosh and P. K. Roy, "Application of earthworm optimization algorithm for solution of optimal power flow," in *2019 International Conference on Opto-Electronics and Applied Optics (Optronix)*, IEEE, Mar 2019.

[145] "Optimal power flow using fuzzy-firefly algorithm," in *2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, IEEE, Oct 2018.

[146] F. Bouffard, F. D. Galiana, and J. M. Arroyo, "Umbrella contingencies in security-constrained optimal power flow," in *15th Power systems computation conference, PSCC*, vol. 5, 2005.

[147] S. Eftekharnejad, "Selection of multiple credible contingencies for real time contingency analysis," in *2015 IEEE Power Energy Society General Meeting*, pp. 1–5, July 2015.

[148] H. Bevrani, *Robust Power System Frequency Control*. Power Electronics and Power Systems, Springer US, 2008.

[149] J. Zhao, H.-D. Chiang, H. Li, and P. Ju, "On pv-pq bus type switching logic in power flow computation," in *Proceedings of the 16th power systems computation conference*, vol. 16, p. 7, jul 2008.

[150] HSL, *A collection of Fortran codes for large scale scientific computation*, 2019 (accessed December 5th, 2019). Availabel in: `http://www.hsl.rl.ac.uk/`.

[151] Y. Yuan, X. Wen, and K. Qian, "Preventive/corrective control for voltage stability based on primal-dual interior point method," in *2006 International Conference on Power System Technology*, pp. 1–5, 2006.

[152] Xuelian Liu, Jiwen Li, Hongmei Li, and Hongxia Peng, "Fuzzy modeling and interior point algorithm of multi-objective opf with voltage security margin," in *2005 IEEE/PES Transmission Distribution Conference Exposition: Asia and Pacific*, pp. 1–6, 2005.

[153] Y. Chen, J. Ma, P. Zhang, F. Liu, and S. Mei, "Robust state estimator based on maximum exponential absolute value," *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 1537–1544, 2017.

[154] R. D. Zimmerman and C. E. Murillo-Sánchez, "Matpower," 2019.

[155] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, apr 2005.

[156] ARPA-E, "SCOPF Problem Formulation: Challenge 1," tech. rep., Advanced Research Projects Agency–Energy), 04 2018.

[157] ARPA-E, *Grid Optimization (GO) Competition*, 2019 (accessed December 5th, 2019). Available in: `https://gocompetition.energy.gov`.

[158] R. D. Zimmerman and C. E. Murillo-Sánchez, "Matpower user's manual," 2019.

[159] M. Bazrafshan, K. Baker, and J. Mohammadi, "Computationally efficient solutions for large-scale security-constrained optimal power flow," 2020.

[160] Zhang, *Solving Large Security-Constrained Optimal Power Flow for Power Grid Planning and Operations*. PhD thesis, Case Western Reserve University, 2020.

[161] S. Huang and V. Dinavahi, "Real-time contingency analysis on massively parallel architectures with compensation method," *IEEE Access*, vol. 6, pp. 44519–44530, 2018.

[162] X. Su, C. He, T. Liu, and L. Wu, "Full Parallel Power Flow Solution: A GPU-CPU Based Vectorization Parallelization and Sparse Techniques for Newton-Raphson Implementation," *IEEE Transactions on Smart Grid*, vol. PP, no. ii, pp. 1–1, 2019.

[163] M. Bazrafshan, K. Baker, and J. Mohammadi, "Computationally efficient solutions for large-scale security-constrained optimal power flow," *arXiv*, pp. 1–8, 2020.

[164] Z. Huang *et al.*, *High-Performance Computing for Real-Time Grid Analysis and Operation*, pp. 151–188. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[165] L. Balduino *et al.*, "Parallel processing in a cluster of microcomputers with application in contingency analysis," in *2004 IEEE/PES Transmision and Distribution Conference and Exposition: Latin America (IEEE Cat. No. 04EX956)*, pp. 285–290, 2004.

[166] G. Angeline Ezhilarasi *et al.*, "Parallel contingency analysis in a high performance computing environment," in *2009 International Conference on Power Systems*, pp. 1–6, 2009.

[167] W. Gao *et al.*, "Distributed generation placement design and contingency analysis with parallel computing technology," *J. Comput.*, vol. 4, pp. 347–354, 2009.

[168] Z. Huang *et al.*, "Massive contingency analysis with high performance computing," in *2009 IEEE Power Energy Society General Meeting*, pp. 1–8, 2009.

[169] Yousu Chen *et al.*, "Performance evaluation of counter-based dynamic load balancing schemes for massive contingency analysis with different computing environments," in *IEEE PES General Meeting*, pp. 1–6, 2010.

[170] S. Jin *et al.*, "A novel application of parallel betweenness centrality to power grid contingency analysis," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pp. 1–7, 2010.

[171] A. Mittal *et al.*, "Real time contingency analysis for power grids," in *Euro-Par 2011 Parallel Processing* (E. Jeannot, R. Namyst, and J. Roman, eds.), (Berlin, Heidelberg), pp. 303–315, Springer Berlin Heidelberg, 2011.

[172] S. K. Khaitan *et al.*, *Dynamic Load Balancing and Scheduling for Parallel Power System Dynamic Contingency Analysis*, pp. 189–209. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[173] S. K. Khaitan *et al.*, "Parallelizing power system contingency analysis using d programming language," in *2013 IEEE Power Energy Society General Meeting*, pp. 1–5, 2013.

[174] S. K. Khaitan *et al.*, "Proactive task scheduling and stealing in master-slave based load balancing for parallel contingency analysis," *Electric Power Systems Research*, vol. 103, pp. 9 – 15, 2013.

[175] G. Zhou *et al.*, "The static security analysis in power system based on spark cloud computing platform," in *2015 IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA)*, pp. 1–6, 2015.

[176] A. Haas, "Pypardisoproject," 2013.

[177] X. Chen, Y. Wang, and H. Yang, "Nicslu: An adaptive sparse matrix solver for parallel circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 2, pp. 261–274, 2013.