Université de Liège

Faculté des Sciences Appliquées
Institut Montefiore
Département d'Electricité, Electronique et Informatique

Time-domain simulation of large electric power systems using domain-decomposition and parallel processing methods

Petros Aristidou

Liège, Belgium, June 2015

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy (Ph.D.) in Engineering Sciences

Examining Committee

Professor Damien Ernst (*President of Jury*), Université de Liège, Belgium
Professor Christophe Geuzaine, Université de Liège, Belgium
Professor Frédéric Magoulès, École Centrale Paris, France
Professor Sakis Meliopoulos, Georgia Institute of Technology, USA
Professor Luis Rouco, Universidad Pontificia Comillas, Spain
Professor Patricia Rousseaux, Université de Liège, Belgium
Professor Thierry Van Cutsem (*Ph.D. advisor*), FNRS and Université de Liège, Belgium

To my family.

Contents

Acknowledgments

I started writing this manuscript with the “simple” purpose of digesting the work of four and a half years (the duration of my doctoral studies) into approximately 200 pages. Almost immediately, I realized that I was not the only contributor to this work. In the following few paragraphs I will try to acknowledge the people that played a role, smaller or bigger, in making this PhD a reality. I will undoubtedly forget or omit some of them, and I apologize and thank them in advance.

First and foremost I offer my sincerest gratitude to my advisor, Professor Thierry Van Cutsem. When I arrived in Liège in 2010, I had little knowledge of what research is. Over the next years, he devoted much of his time to transfer to me his theoretical and practical understanding of power systems, and to help me develop valuable skills as a researcher and as an academic. He has always encouraged and trusted me to venture on new ideas, even when those diverted from my original research plans. In addition to our academic interests, we also shared a love for photography; on several occasions we would spent our spare time exchanging moments captured on “film” and discussing some new photography equipment or technique. Overall, he has been an exceptional mentor and an outstanding friend, on whom I could always rely for help and guidance.

I wish to express my gratitude to each member of the examining committee, for devoting their time to read this report. During my stay in Liège, Professors Patricia Rousseaux, Christophe Geuzaine, and Damien Ernst have offered concrete support both as members of my dissertation committee but also through our enriching discussions. A special thanks to Professor Costas Vournas who was the first one to introduce me to the world of power system dynamics and has supported and advised me over the years. I would also like to thank Dr. Mevludin Glavic for the insightful discussions during our coffee breaks outside the department entrance, but also for transferring his experience to help me make some important decisions in my life. I wish to thank Professor Xavier Guillaud, his support and feedback on my work have been valuable for my PhD. Many thanks to Simon Lebeau (TransÉnergie division, Hydro-Québec) and Patrick Panciatici (RTE) for providing a valuable input to my work based on real engineering problems. Upon arriving in Liège, I was welcomed by Professor

Mania Pavella, who showed great interest in my academic development and my general well-being, and I'm thankful for that. I am also really grateful to the Bodossaki Foundation and Mr. Sotiri Laganopoulo, for the invaluable support and guidance they offered.

Throughout the process, I have greatly benefited from the work and feedback of Professor Van Cutsem's former and current students and visitors. I would like to thank Dr. Davide Fabozzi for his support during my first two years in Liège. He has proved to be a valuable source of information (on technical and non-technical topics alike) and a dear friend. I tried to match his enthusiasm and provide the same level of support to my junior PhD colleagues Lampro Papangeli and Hamid Soleimani. Their collaboration and friendship made the last two years more enjoyable. Special thanks to Frédéric Plumier, for being a close friend and a great colleague. He patiently allowed me to "torture" his mother language so I can practice my French, and introduced me into his family as the godfather of Antoine; for that, I will always be indebted. Also, I would like to thank Professor Gustavo Valverde, from whom I have learned many lessons, a superb researcher and most importantly an exceptional friend, and of course to his lovely wife, Rebecca. My thanks to all the researchers who briefly joined the Liège group, amongst which, Benjamin Saive, Dr. Tilman Weckesser, Dr. Spyros Chatzivasileiadis, and Theodoros Kyriakidis. Your support and friendship are cherished. I am also grateful to Dr. Efthymios Karangelos and Panagiotis Andrianesis, for promptly offering their support and advice whenever needed.

I cannot list everyone here, but I am grateful to all my friends, the ones from Cyprus and Greece, as well as the people I befriended during these last years in Belgium. My thoughts are with you, wherever you are.

The last thanks go to my family, my sister Maria with whom we spent a big amount of time consulting and listening to each other during our parallel PhD journeys; my sister Angela, an inspiration and beacon to my academic ventures, and her husband Alex, for being my unofficial academic advisors over the years, and for bringing to life Stella, a shinning star of joy and happiness in our family. A warm thank you to Dafni, for all her love and support that made these years more enjoyable and gave me the strength to see this journey through.

The end of my doctoral studies in Belgium signals 12 years since I left my parents' home in Pissouri, Cyprus. Nevertheless, my parents Christos and Stella, have been by my side every single day since then; through my days in the military service, my diploma studies in Athens, and my doctoral studies in Liège. Their love and support (material and emotional) have allowed me to constantly leap forward in new endeavors without any fear, as I always know that "they have my back". I also want to thank my beloved grandmother Angeliki for her heart-warming discussions over the phone and the delicious food she prepared for me at every opportunity.

Thank you all,

Petros

Abstract

Dynamic simulation studies are used to analyze the behavior of power systems after a disturbance has occurred. Over the last decades, they have become indispensable to anyone involved in power system planning, control, operation, and security. Transmission system operators depend on fast and accurate dynamic simulations to train their personnel, analyze large sets of scenarios, assess the security of the network in real-time, and schedule the day-ahead operation. In addition, those designing future power systems depend on dynamic simulations to evaluate proposed reinforcements, whether these involve adding new transmission lines, increasing renewable energy sources, or implementing new control schemes.

Even though almost all computers are now parallel, power system dynamic simulators are still based on monolithic, circuit-based, single-process algorithms. This is mainly due to legacy code, written in the 80's, that is still today in the core of the most important commercial tools and does not allow them to fully exploit the parallel computational resources of modern computers.

In this thesis, two parallel algorithms belonging to the family of Domain Decomposition Methods are developed to tackle the computational complexity of power system dynamic simulations. The first proposed algorithm is focused on accelerating the dynamic simulation of large interconnected systems; while, the second algorithm aims at accelerating dynamic simulations of large combined transmission and distribution systems.

Both proposed algorithms employ non-overlapping decomposition schemes to partition the power system model and expose parallelism. Then, "divide-and-conquer" techniques are utilized and adapted to exploit this parallelism. These algorithms allow the full usage of parallel processing resources available in modern, inexpensive, multi-core machines to accelerate the dynamic simulations. In addition, some numerical acceleration techniques are proposed to further speed-up the parallel simulations with little or no impact on accuracy.

All the techniques proposed and developed in this thesis have been thoroughly tested on academic systems, a large real-life system, and a realistic system representative of the continental European synchronous grid. The investigations were performed on a large multi-core machine, set up for the needs of this work, as well as on two multi-core laptops computers.

Nomenclature

Abbreviations

ADN	Active Distribution Network
API	Application Programming Interface
ASRT	Automatic Shunt Reactor Tripping
BDF	Backward Differentiation Formulae
BEM	Backward Euler Method
COI	Center of Inertia
DAE	Differential-Algebraic Equation
DCTL	Dscrete controller
DDM	Domain Decomposition Method
DG	Distributed Generator
DN	Distribution Network
DNV	Distribution Network Voltage
DSA	Dynamic Security Assessment
EMA	Exponential Moving Average
EMT	ElectroMagnetic Transients
GPU	Graphics Processing Unit
HPC	High-Performance Computing
HQ	Hydro-Québec
IN	Inexact Newton
IVP	Initial Value Problem
LTC	Load Tap Changer
LVFRT	Low Voltage and Fault Ride Through
NUMA	Non-Uniform Memory Access
ODE	Ordinary Differential Equation
OHC	OverHead Cost
OXL	OvereXcitation Limiters
PV	PhotoVoltaic

RHS	Right-Hand Side
T&D	Transmission and Distribution
TN	Transmission Network
TSO	Transmission System Operator
UMA	Uniform Memory Access
VDHN	Very DisHonest Newton
WR	Waveform Relaxation

Mathematical Symbols

$\boldsymbol{D}$	matrix that includes the real and imaginary parts of the bus admittance matrix
$\mathbf{\Gamma}$	diagonal matrix with $(\mathbf{\Gamma})_{\ell\ell}$ equal to 1 if the l-th equation is differential and 0 if it is algebraic
$\mathbf{\Gamma}_C$, $\mathbf{\Gamma}_{Si}$	projection of $\mathbf{\Gamma}$ on the Central or i-th Satellite sub-domain
$\mathbf{\Gamma}_i$	projection of $\mathbf{\Gamma}$ on the i-th injector sub-domain
$\boldsymbol{A}_i$	Jacobian matrix of injector i
$\boldsymbol{B}_i$	matrix of sensitivities of injectors equations to voltages in injector i
$\boldsymbol{C}_i$	trivial matrix with zeros and ones linking injector i currents to network equations
$\boldsymbol{I}$	sub-vector of $\boldsymbol{x}$ containing the bus currents
$\boldsymbol{J}$	Jacobian matrix of both network and injectors
$\boldsymbol{V}$	vector of rectangular components of bus voltages
$\boldsymbol{x}$	state vector containing the differential and algebraic variables
$\boldsymbol{x}_C$, $\boldsymbol{x}_{Si}$	projection of $\boldsymbol{x}$ on the Central or i-th Satellite sub-domain
$\boldsymbol{x}_i$	projection of $\boldsymbol{x}$ on the i-th injector sub-domain
L	number of Satellite sub-domains
M	number of parallel workers
N_C, N_{Si}	number of injectors attached on the Central or i-th Satellite sub-domain network
T_1^*	run-time of a program with one worker using the fastest (or a very fast) sequential algorithm
T_1	the time an algorithm takes to run in sequential execution ($M = 1$)
T_∞	the time an algorithm takes on an ideal machine with an infinite number of parallel workers ($M = \infty$)
T_P	percentage of time spent in parallel execution
T_S	percentage of time spent in sequential execution

CHAPTER 1

Introduction

1.1 Motivation

Dynamic simulations under the phasor approximation are routinely used throughout the world for the purpose of checking the response of electric power systems to large disturbances. Over the last decades, they have become indispensable to anyone involved in the planning, design, operation, and security of power systems. Power system operators depend on fast and accurate dynamic simulations to train operators, analyze large sets of scenarios, assess the dynamic security of the network in real-time, or schedule the day ahead operation. On the other hand, those designing future power systems depend on dynamic simulations to evaluate the proposed changes, whether these involve adding new transmission lines, increasing renewable energy sources, implementing new control schemes, or decommissioning old power plants.

Such simulations require solving a large set of nonlinear, stiff, hybrid, Differential-Algebraic Equations (DAEs) that describe the physical dynamic characteristics, interactions, and control schemes of the system. A large interconnected transmission or a detailed transmission and distribution system may involve hundreds of thousands of such equations whose dynamics span over very different time scales and undergo many discrete transitions imposed by limiters, switching devices, etc. Consequently, dynamic simulations are challenging to perform, computationally intensive and can easily push any given computer to its limits.

In applications targeting the real-time monitoring and security of the system, for example Dynamic Security Assessment (DSA), the speed of simulation is a critical factor. In the remaining applications, speed is not critical but desired as it increases productivity. This is the main reason why energy management systems often resort to faster, static simulations.

However, the operation of non-expandable grids closer to their stability limits and the unplanned generation patterns stemming from renewable energy sources require dynamic studies. Furthermore, under the pressure of electricity markets and with the support of active demand response, it is likely that system security will be more and more guaranteed by emergency controls responding to the disturbance. Thus, checking the sequence of events

that take place after the initiating disturbance is crucial; a task for which the static calculation of the operating point in a guessed final configuration is inappropriate.

All modern computers are now parallel. Even the smallest ones, such as mobile phones, offer at least one parallel feature, such as vector instructions, multithreaded cores, multicore processors, multiple processors, graphic processing units, or parallel co-processors. The advent of parallel processing represents a revolutionary opportunity for power system dynamic simulation software.

Unfortunately, the traditional approach to perform these simulations is based on monolithic, circuit-based, single-process schemes. This is mainly due to legacy code, written in the 80's, that is still today at the heart of the most important commercial tools for power system dynamic simulations. Many of these programs are serial not because it was natural to solve the problem serially, but because the programming tools demanded it and the programmers were trained to think that way. This approach hinders the simulation performance, decreasing productivity and increasing the overall cost.

Among the commercial software, PowerWorld, Digsilent Power Factory, ETAP (Electrical Transient Analyzer Program), PSLF, EuroStag, PSS/E, Simpow, and CYME are well-known simulators. To our knowledge, none of these widely used software offers multithreaded dynamic simulations yet. This means that they run only on one core of one processor, and therefore far from fully utilize the whole power of the new parallel computers. On the other hand, existing commercial parallel simulators (such as Opal-RT ePHASORsim) are based on specialized parallel computers. Finally, due to their closed source, commercial software do not provide full flexibility for experimentation and prototyping.

1.2 Power system modeling

Power system dynamic simulations fall in basically two categories: ElectroMagnetic Transients (EMT) and quasi-sinusoidal (or phasor mode) approximation. In the former, fast electromagnetic transients are simulated and in steady state, voltages and currents evolve sinusoidally with time at a frequency close to the nominal value (50 or 60 Hz). The network itself is modeled through differential equations relative to its inductors and capacitors. On the other hand, in the quasi-sinusoidal (or phasor mode) approximation, the network is represented through algebraic equations corresponding to sinusoidal regime. During transients, all phasors vary with time while in steady-state they take on constant values [Kun94, MBB08]. The dynamic model describes how the phasors evolve with time.

The work presented in this thesis falls in the second category. It is the one commonly used in stability studies, where the simulated time interval can extend up to several minutes, if not more. Thus, the models presented hereafter are based on the following assumptions:

1. power system components are modeled by a set of nonlinear DAEs,
2. AC devices operate in three-phase balanced fundamental frequency, and

3. the analyzed dynamics have time constants of tenths to tens of seconds or equivalently 0.1 to 10 Hz.

1.2.1 Model overview

An electric power system, under the quasi-sinusoidal approximation, can be described in compact form by the following DAE Initial Value Problem (IVP):

$$\mathbf{0} = \boldsymbol{\Psi}(\boldsymbol{x}, \boldsymbol{V}) \tag{1.1a}$$

$$\boldsymbol{\Gamma}\dot{\boldsymbol{x}} = \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{V}) \tag{1.1b}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0, \boldsymbol{V}(t_0) = \boldsymbol{V}_0 \tag{1.1c}$$

where $\boldsymbol{V}$ is the vector of voltages through the network and $\boldsymbol{x}$ is the state vector containing the remaining (except voltages) differential and algebraic variables of the system. Furthermore, $\boldsymbol{\Gamma}$ is a diagonal matrix with:

$$(\boldsymbol{\Gamma})_{\ell\ell} = \begin{cases} 0 & \text{if } \ell\text{-th equation is algebraic} \\ 1 & \text{if } \ell\text{-th equation is differential} \end{cases} \tag{1.2}$$

The algebraic Eq. 1.1a describes the network and can be rewritten as:

$$\mathbf{0} = \boldsymbol{D}\boldsymbol{V} - \boldsymbol{I} \triangleq \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{V}) \tag{1.3}$$

where $\boldsymbol{D}$ includes the real and imaginary parts of the bus admittance matrix and $\boldsymbol{I}$ is a sub-vector of $\boldsymbol{x}$ containing the bus currents [Fab12].

The initial voltages $\boldsymbol{V}_0$ and currents $\boldsymbol{I}_0$ are usually obtained by performing a power-flow computation of the static power system model or are received from a state estimation software. Next, the remaining DAE states $\boldsymbol{x}_0$ are computed through an initialization procedure.

Equation 1.1b describes the remaining DAEs of the system including the dynamics of generating units, their controls, dynamic loads, and other devices. Together these equations form a complete mathematical model of the system, which can be solved numerically to simulate the system behavior.

1.2.2 Numerical integration methods

The analytical solution of Eqs. 1.1 is not generally possible [MBB08]. Therefore, a numerical solution consisting of a series of values $(\boldsymbol{V}_1, \boldsymbol{x}_1)$, $(\boldsymbol{V}_2, \boldsymbol{x}_2)$, ..., $(\boldsymbol{V}_k, \boldsymbol{x}_k)$ that satisfies the equations (1.1) at the time instants t_1, t_2, ..., t_k must be found. This requires the use of a numerical integration formula that calculates the value of $(\boldsymbol{V}_{k+1}, \boldsymbol{x}_{k+1})$ knowing all the previous values.

Problem (1.1) is characterized as a semi-explicit DAE of index-1 (or Heisenberg index-1) [BCP95]. The differential *index* of the system is the minimal number of analytical differentiations needed such that the DAE equations can be transformed by algebraic manipulations into a system of explicit Ordinary Differential Equation (ODE) system [BCP95]. In power

system transient stability simulations the index-1 condition is assured for most of practical cases except for some operating conditions close to voltage collapse [LB93]. The theory behind numerical methods for general DAE systems is very complex. Fortunately, semi-explicit index-1 systems can be directly discretized using classical numerical integration methods for ODEs, while adapting the solution algorithm to take take into account the algebraic constraints [BCP95]. These integration methods fall into two general categories: *explicit* or *implicit*, the latter of which will be detailed in the sequel.

Power system dynamic simulation models involve *stiff* DAEs. A stiff problem is one in which the underlying physical process contains components evolving on widely separated time scales [BCP95], or equivalently the dynamics of some part of the process are very fast compared to the interval over which the whole process is studied. In linear systems, stiffness is measured by the ratio of the largest to the smallest eigenvalue. Practically, this feature of system (1.1) requires integration methods with properties such as A-stability and stiff decay (or L-stability) [BCP95, MBB08].

1.2.2.1 Implicit methods

Ideally, a numerical integration method should mimic all properties of the differential problem, for all types of problems. However, this is not possible. Thus, the integration method used should captures at least the essential properties of a class of problems. In this section, the discussion is concentrated on implicit integration methods because they can be effectively used to solve stiff DAEs [BCP95].

Let us consider the following ODE:

$$\dot{y} = c\left(t, y\right) \tag{1.4}$$

where the dependence to the time variable t will be omitted hereon.

Many frequently used implicit integration formulas can be written in the general form [BCP95]:

$$y_{k+1} = \beta_k + h\beta_0 c\left(y_{k+1}\right) \tag{1.5}$$

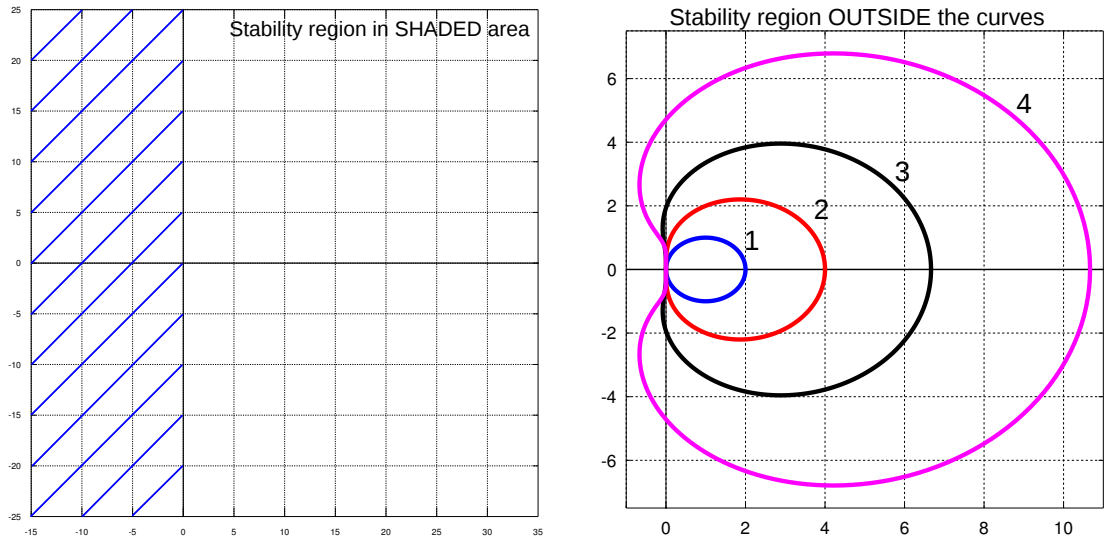
where h is the integration step length, β_0 is a coefficient that depends on the actual integration method, $c\left(y_{k+1}\right)$ is the right-hand side of the differential equation calculated at the value y_{k+1}, and:

$$\beta_k = y_k + \sum_j a_j c(y_{k+1-j}) \tag{1.6}$$

is a coefficient depending on all the previous steps [BCP95, MBB08].

As mentioned previously, one of the desired properties of integration formulae is A-stability. This property can be defined using the scalar test equation:

$$\dot{y} = \lambda y \tag{1.7}$$



(a) Trapezoidal method (b) BDF orders 1-4

Figure 1.1: Area of absolute stability

where $\lambda \epsilon \mathbb{C}$ is a complex constant. For an integration to be A-stable, it is required that whenever the exact solution of (1.7) is stable (i.e. λ has negative real part), so is the simulated one whatever the step size h. In other words, a stable system is always simulated as stable.

In addition, it is also desirable that, whenever the exact solution of (1.7) is unstable (i.e. λ has positive real part), so is the simulated one. That is, an unstable system is always simulated as unstable. The area for which this does not hold true is called area of hyper-stability [BCP95].

The most well-known A-stable method is the Trapezoidal rule method. The latter is formulated as:

$$y_{k+1} = y_k + \frac{h}{2} c\left(y_{k+1}\right) + \frac{h}{2} c\left(y_k\right) \tag{1.8}$$

This method has been widely used in power system applications, mainly because it is *symmetrically A-stable*. That is, it is A-stable and doesn't include any region of hyper-stability, as shown in Fig. 1.1a [SCM98].

However, the most popular methods for *stiff* problems are the Backward Differentiation Formulae (BDF). Their distinguishing feature is that $c(y)$ is evaluated only at the right end of the current step. This leads to formulae with the stiff decay (or L-stability) property. Let us generalize the test equation to:

$$\dot{y} = \lambda(y - c(t)) \tag{1.9}$$

where $c(t)$ is a bounded function. Assuming that the system is stable, an integration method has stiff decay if, for a given $t_k > 0$:

$$|y_k - c(t_k)| \to 0 \quad \text{as} \quad h\Re e(\lambda) \to -\infty \tag{1.10}$$

The practical advantage of stiff decay methods lies in their ability to skip fine-level (or fast varying) dynamics and still maintain a good description of the solution on a coarse level [BCP95]. Thus, the choice of the time-step size is not dictated strictly by accuracy, but by what should be retained in the final response [Fab12]. Conversely, integration methods without stiff decay (such as the Trapezoidal method) need to be used with small enough time steps even if only the coarse behavior of the solution is sought, otherwise errors propagate and numerical oscillations occur [GSD+03].

The simplest BDF method is the Backward Euler Method (BEM), formulated as:

$$y_{k+1} = y_k + hc\left(y_{k+1}\right) \tag{1.11}$$

A very important feature of BEM is that it allows to integrate over a discontinuity. In the corresponding Eq. 1.11, if the discontinuity takes place at time t_k, only the derivative at time t_{k+1} is used to compute y_{k+1}. Therefore, this scheme can be used for the time step that immediately follows a discontinuity.

Another well-known member method of this family is the second-order BDF (BDF-2), formulated as:

$$y_{k+1} = \frac{4}{3}y_k - \frac{1}{3}y_{k-1} + \frac{2}{3}h\,c\left(y_{k+1}\right) \tag{1.12}$$

As seen in Fig. 1.1b, BDFs of order larger than two are not A-stable. Moreover, all BDFs have a region of hyper-stability (although it decreases when the order increases).

Further analysis on the properties and applications of ODE methods can be found in dedicated references, such as [Gea71, BCP95, HW96]. More specific and recent developments on numerical integration methods used in power system applications can be found in [SCM08, MBB08, Mil10, ES13]. The integration formula of choice in this thesis is the second-order BDF, both for the benchmark and the proposed methods. This formula is initialized by BEM, which is also used in case of discontinuities.

1.2.3 Time-step selection

The integration step h is usually not constant throughout the simulation. In the short-term period following a disturbance, the fast dynamics dominate the system behavior and a small time-step size is needed to simulate them correctly. In the long-term period, the time-step size may increase, if the integration method allows it, since the period is dominated by slow dynamics. Finally, a new event could initiate again fast dynamics, leading to a decrease of the time-step size.

Algorithms that can change the step size during the simulation are called “variable time step algorithms”. There exist two main strategies for selecting the time-step size: according to desired *error* or *effort*. A popular way to achieve the desired precision is through the estimation of the Local Truncation Error (LTE) [Gea71, BCP95]. The LTE for a method of order p and time-step size h is in the order of h^p and can be estimated directly using the current and previous values of the simulation [BCP95]. The error estimate can be used to

decide whether to accept the results of the current step or to redo with a smaller step size. Alternatively, it can be used to select the next time-step size to be as big as possible within the LTE tolerance.

The second strategy consists in varying the time-step size according to the estimated computational effort, based on the assumption that the computational effort varies with the step size. This can be done either by relying on the magnitude of the mismatch vector [FV09], or on the number of Newton iterations [Fab12], originally proposed in [Sto79], for error estimation. In this work, the latter is followed.

Although variable time step algorithms allow increasing the step size (provided that the error estimation remains within the desired bounds), they do so assuming a continuous, non-discrete, system of DAEs. However, power system models are hybrid systems[1]: during the simulation they undergo discrete transitions. The occurrence of these discrete changes should be factored in the selection of the time-step size as discussed in the following section.

1.2.4 Treatment of discrete events

The hybrid nature of power system models [HP00] makes them more problematic to handle as the discrete changes force discontinuities in Eqs. 1.1. In the general case, whether the solver uses constant or variable step size, those instants where discrete variables change do not coincide with a simulation time step. From a strict mathematical viewpoint, it is not correct to integrate over these discontinuities; in fact, all numerical integration methods are based on polynomial expansions which cannot reproduce discontinuities [Fab12].

One option is to identify the time t^* of the discrete event (e.g. using zero-crossing function analysis) and adjust the step size to coincide with the upcoming event. Next, Eqs. 1.1 are updated and the resulting discontinuity is solved appropriately. Then, the solution proceeds with next time step [Cel06]. This mechanism leads to accurate simulations but imposes a limit on the time-step size allowed and a reduction of the average step size. Moreover, it increases the computational burden for the identification of discrete event timings (e.g. solution of zero-crossing functions).

On the other hand, if the step size is not reduced, the discrete event and the update of equations are shifted in time. This may cause variables to converge to wrong values, or even not converge, if some modeling and solving precautions are not taken. An appropriate *ex post* treatment of these shifted discrete events has been proposed in [FCPV11]. In brief, the time step $t \rightarrow t + h$ is computed and the solution is checked for any occurring discrete transitions within this interval. If it is detected that a discrete change has occurred, Eqs. 1.1 are updated accordingly and the same time step is computed again. This procedure is repeated until no discrete transitions occur any longer or a maximum number of jumps is reached. Then, the simulation proceeds with a new step. This approach is followed in this work.

[1] The term "hybrid system" shouldn't be confused with hybrid transient simulations that indicates tools that integrate together EMT and phasor mode models

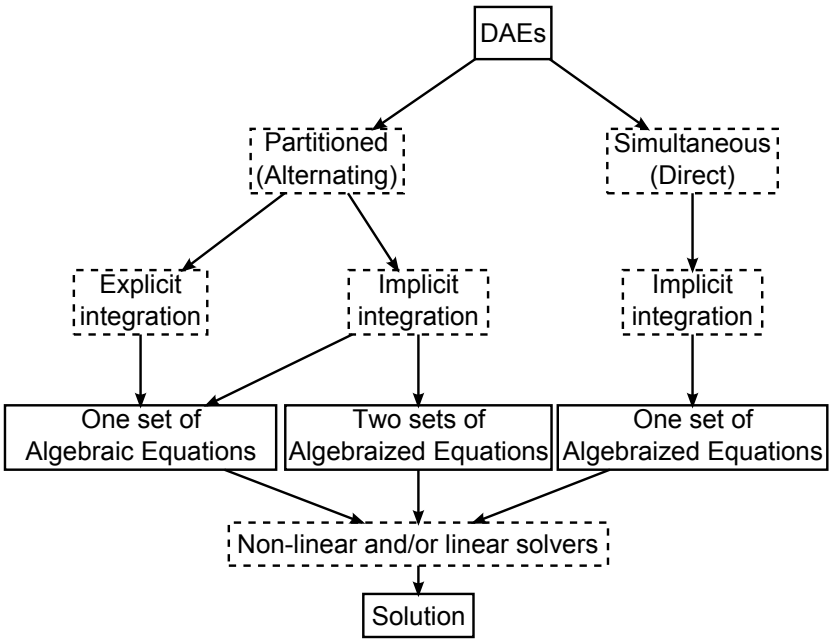


Figure 1.2: Categories of solution approaches for dynamic simulation

1.2.5 Dealing with algebraic and differential equations

The schemes to solve (1.1) are characterized by:

- the way that algebraic and differential equations are interfaced: partitioned or simultaneous
- the type of integration methods used: explicit or implicit.

Thus, the solution approaches can be divided into two main categories [DS72, Sto79, MBB08, FMT13] as shown in Fig. 1.2.

1.2.5.1 Partitioned approach

In the partitioned approach, the differential and the algebraic equations in (1.1) are solved independently. At each step of the numerical integration procedure the partitioned scheme alternates between the solutions of respectively the differential and the algebraic equations. The latter are usually solved with a Newton method (see Appendix A) while the differential equations are usually solved by functional iterations [MBB08]. However, the convergence of this scheme significantly limits the allowed time-step size [BCP95].

The partitioned scheme is attractive for short-term simulations. It is flexible, easy to organize and allows a number of simplifications to be introduced in order to speed up the solution [MBB08]. For these reasons, quite a number of dynamic simulation programs are still based on partitioned solution methods.

However, this alternating procedure introduces a "delay" between these variables. That is, when computing the differential variables, the algebraic are "frozen" and vice-versa. Therefore, unless care is taken, this process can lead to numerical instabilities or decreased performance [Mil10].

1.2.5.2 Simultaneous approach

Unlike the partitioned approach, the simultaneous one combines the algebraized differential and the algebraic equations of (1.1) into one set of nonlinear, algebraic equations. This approach is used in conjunction with implicit numerical methods that require, at each time step, the solution of a set of nonlinear equations. The solution is generally obtained through a Newton's method (see Appendix A), which requires iteratively computing and factorizing a Jacobian matrix [Mil10].

Using (1.5) to algebraize Eqs. 1.1, the following equations are obtained:

$$\boldsymbol{g}\left(\boldsymbol{x}_{k+1}, \boldsymbol{V}_{k+1}\right) = \boldsymbol{\Psi}\left(\boldsymbol{x}_{k+1}, \boldsymbol{V}_{k+1}\right) = \mathbf{0} \tag{1.13a}$$

$$\boldsymbol{f}\left(\boldsymbol{x}_{k+1}, \boldsymbol{V}_{k+1}\right) = \boldsymbol{\Phi}\left(\boldsymbol{x}_{k+1}, \boldsymbol{V}_{k+1}\right) - \frac{1}{h\beta_0}\boldsymbol{\Gamma}\left(\boldsymbol{x}_{k+1} - \boldsymbol{\beta}_k\right) = \mathbf{0} \tag{1.13b}$$

where $\boldsymbol{\beta}_k$ is a column vector containing the values β_k of Eq. 1.6.

The formula for the l-th iteration of Newton's method for the k-th time step is given as follows:

$$\begin{bmatrix} \boldsymbol{V}_{k+1}^{(l+1)} \\ \boldsymbol{x}_{k+1}^{(l+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{V}_{k+1}^{(l)} \\ \boldsymbol{x}_{k+1}^{(l)} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Psi}_{\boldsymbol{V}} & \boldsymbol{\Psi}_{\boldsymbol{x}} \\ \boldsymbol{\Phi}_{\boldsymbol{V}} & \boldsymbol{\Phi}_{\boldsymbol{x}} - \frac{1}{h\beta_0}\boldsymbol{\Gamma} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{g}\left(\boldsymbol{x}_{k+1}^{(l)}, \boldsymbol{V}_{k+1}^{(l)}\right) \\ \boldsymbol{f}\left(\boldsymbol{x}_{k+1}^{(l)}, \boldsymbol{V}_{k+1}^{(l)}\right) \end{bmatrix} \tag{1.14}$$

where $\boldsymbol{\Phi}_{\boldsymbol{x}} = \partial\boldsymbol{\Phi}/\partial\boldsymbol{x}$, $\boldsymbol{\Phi}_{\boldsymbol{V}} = \partial\boldsymbol{\Phi}/\partial\boldsymbol{V}$, $\boldsymbol{\Psi}_{\boldsymbol{x}} = \partial\boldsymbol{\Psi}/\partial\boldsymbol{x}$, and $\boldsymbol{\Psi}_{\boldsymbol{V}} = \partial\boldsymbol{\Psi}/\partial\boldsymbol{V}$ are the Jacobian sub-matrices. The Jacobian matrix is sparse, so computer programs that simulate large systems do not explicitly invert this matrix. Instead, Eq. 1.14 is reformulated as:

$$\begin{bmatrix} \boldsymbol{\Psi}_{\boldsymbol{V}} & \boldsymbol{\Psi}_{\boldsymbol{x}} \\ \boldsymbol{\Phi}_{\boldsymbol{V}} & \boldsymbol{\Phi}_{\boldsymbol{x}} - \frac{1}{h\beta_0}\boldsymbol{\Gamma} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{V}_{k+1}^{(l)} \\ \Delta\boldsymbol{x}_{k+1}^{(l)} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{g}\left(\boldsymbol{x}_{k+1}^{(l)}, \boldsymbol{V}_{k+1}^{(l)}\right) \\ \boldsymbol{f}\left(\boldsymbol{x}_{k+1}^{(l)}, \boldsymbol{V}_{k+1}^{(l)}\right) \end{bmatrix} \tag{1.15}$$

and solved using a sparse linear solver.

The simultaneous solution methods are especially attractive for simulations that cover a long time period [MBB08, Mil10]. Newton's method, together with a stiff decay integration formulae, allow the integration step size to be increased when the changes in the variables are not very steep. The so-called Very DisHonest Newton (VDHN) method, described in Appendix A, can be used to further speed up the calculations. Finally, interfacing the algebraic and differential equations is not a problem as both the algebraic and differential variables are updated together [MBB08].

The convergence criteria used to stop the Newton iterations significantly impacts the overall efficiency of the solution scheme. The vector in the Right-Hand Side (RHS) of (1.15)

is usually referred to as "mismatch vector" (of the $\boldsymbol{f}$ and $\boldsymbol{g}$ equations, respectively). To ensure that all equations are solved within the specified tolerance, the infinite norm (i.e. the largest component magnitude) of each mismatch vector should be brought below some value. Thus, the Newton iterations are stopped if:

$$\left\| \boldsymbol{g}\left(\boldsymbol{x}_{k+1}^{(l)}, \boldsymbol{V}_{k+1}^{(l)}\right) \right\|_{\infty} < \epsilon_g \tag{1.16a}$$

$$\left\| \boldsymbol{f}\left(\boldsymbol{x}_{k+1}^{(l)}, \boldsymbol{V}_{k+1}^{(l)}\right) \right\|_{\infty} < \epsilon_f \tag{1.16b}$$

In power system dynamic simulations it is usual that the network variables and parameters are set in the *per-unit system* [MBB08]. That is, a base power is selected (S_{base}) for the entire system and using the base voltage levels (usually the nominal voltage of a bus), all the network parameters and variables are scaled accordingly. Thus, for the network equations $\boldsymbol{g}$, the choice of ϵ_g is rather easy. For the remaining equations, however, it may be difficult to choose an appropriate ϵ_f value as they include a variety of models and controls for which the solver does not know whether a mismatch is "negligible" or not. This issue can be resolved by checking the normalized corrections instead. Thus, Eq. 1.16b is replaced by:

$$\left| \left(\Delta \boldsymbol{x}_{k+1}^{(l)}\right)_i \right| < max\left(\epsilon_{fabs}, \epsilon_{frel} \left| \left(\boldsymbol{x}_{k+1}^{(l)}\right)_i \right|\right) \quad i = 1, ..., dim(\boldsymbol{x}) \tag{1.17}$$

where the *relative* correction is checked against ϵ_{frel}. Of course, when $\left|\boldsymbol{x}_{k+1}^{(l)}\right|$ becomes small, the absolute correction should be checked against ϵ_{fabs} to avoid numerical exceptions. The price to pay is the computation of $\Delta \boldsymbol{x}_{k+1}^{(l)}$, which requires making *at least one iteration* before deciding on the convergence [Fab12, FCHV13].

1.2.6 System reference frame

The network equations are usually expressed in rectangular form in the system's rotating reference frame, that is $\boldsymbol{V} = (\boldsymbol{V}_x, \boldsymbol{V}_y)$ and $\boldsymbol{I} = (\boldsymbol{I}_x, \boldsymbol{I}_y)$. The selection of the speed (ω_{ref}) at which the $x-y$ reference axes rotate is arbitrary; hence, it can be taken for convenience of the computations. Standard practice for short-term (e.g. transient stability) analysis is to choose the nominal angular frequency $\omega_0 = 2\pi f_0$, where f_o is the nominal system frequency. However, this reference frame suffers from a major drawback in long-term studies. After a disturbance that modifies the power balance, the system settles at a new angular frequency $\omega \neq \omega_0$. Thus, when projected to axes rotating at the speed ω_0, all phasors rotate at the angular speed $\omega - \omega_0$. Consequently, even though the system settles at a new equilibrium, the current and voltage components ($\boldsymbol{V}_x, \boldsymbol{V}_y, \boldsymbol{I}_x$, and $\boldsymbol{I}_y$) oscillate with a period $T = 2\pi|\omega - \omega_0|^{-1}$. This behavior can trigger more solutions of the system and impose the time step to remain small compared to T, in order to track the oscillations and avoid numerical instability.

This problem can be partially solved by adopting the Center-Of-Inertia (COI) reference frame initially proposed in [TM72]. The underlying idea is to link the reference axes to the

rotor motion of synchronous machines. The COI speed is defined as:

$$\omega_{coi} = \frac{1}{M_T}\sum_{i=1}^{m} M_i\omega_i \tag{1.18}$$

where M_i and ω_i are respectively the inertia and rotor speed of the i-th synchronous machine and $M_T = \sum_{i=1}^{m} M_i$ is the total inertia. Thus, if the system settles at a new equilibrium with angular frequency ω, all machines and the $x-y$ reference axes rotate at the same angular speed and the current and voltage components become constant.

However, the exact COI reference frame is computationally expensive as it introduces Eq. 1.18 which couples the rotor speeds of all synchronous machines in the network, while the motion equation of each machine involves ω_{coi}. If the simulation is performed with the simultaneous approach, it adds a *dense* row and column to the Jacobian of (1.15). Moreover, this coupling impedes the decomposition of the system as it introduces a coupling between components all over the system. To alleviate this dependency, it has been proposed in [FV09] to use the ω_{coi} value *of the previous time instant*. This value can be computed explicitly as it refers to past, already known speed values. The advantages of the COI reference frame are also preserved, because a slightly delayed COI angle is as good as the exact COI in making the current and voltage components very little dependent on the system frequency [FV09]. This reference frame is used in this work.

1.3 Description of power system models used in this work

In this section the power system models used in the following chapters are briefly presented. Their diagrams are presented in Appendix B.

1.3.1 Nordic system

This is a variant of the so-called Nordic32 test system [VP13]. Its one-line diagram is presented in Fig. B.2. It is a fictitious system inspired of the Swedish system in a past configuration. The test system includes 54 generator and transmission buses and 80 branches. When including the distribution buses and transformers, there are a total of 77 buses and 105 branches, respectively. Furthermore, 20 synchronous machines are represented along with generic excitation systems, voltage regulators, power system stabilizers, speed governors, and turbine models. Finally, 23 dynamically modeled loads are included, attached to the distribution buses. The model sums to 750 DAEs.

Two variants of this model will be used in Chapter 5 to simulate combined transmission and distribution systems. In the *first variant*, the Nordic model is expanded with 146 Distribution Networks (DNs) that replace the aggregated distribution loads. The model and data of each DN (shown in Fig. B.1) were taken from [Ish08] and scaled to match the original loads seen by the Transmission Network (TN). Multiple DNs were used to match the original loads,

taking into account the nominal power of the TN-DN transformers. Each one of the 146 DNs is connected to the TN through two parallel transformers equipped with Load Tap Changer (LTC) devices. Each DN includes 100 buses, one distribution voltage regulator equipped with LTC, three PhotoVoltaic (PV) units [WEC14], three type-2, two type-3 Wind Turbines (WTs) [EKM+11], and 133 dynamically modeled loads, namely small induction machines and exponential loads. In total, the combined transmission and distribution system includes 14653 buses, 15994 branches, 23 large synchronous machines, 438 PVs, 730 WTs, and 19419 dynamically modeled loads. The resulting model has 143462 differential-algebraic states. The one-line diagram is sketched in Fig. B.3.

In the *second variant*, six of the aggregated DN loads in the *Central* area are replaced by 40 detailed Active Distribution Networks (ADNs), each equipped with the Distribution Network Voltage (DNV) controller described in [VV13]. Each DN is a replica of the same medium-voltage distribution system, whose one-line diagram is shown in Fig. B.4. It consists of eight 11-kV feeders all directly connected to the TN-DN transformer, involving 76 buses and 75 branches. The various DNs were scaled to match the original (aggregate) load powers, while respecting the nominal values of the TN-DN transformers and other DN equipment. The combined transmission and distribution model includes 3108 buses, 20 large and 520 small synchronous generators, 600 induction motors, 360 type-3 WTs [EKM+11], 2136 voltage-dependent loads, and 56 LTC-equipped transformers. The resulting model has 36504 differential-algebraic states. The one-line diagram is sketched in Fig. B.5.

1.3.2 Hydro-Québec system

Hydro-Québec (HQ) is the Transmission System Operator (TSO) of the Québec province, Canada. This real-life model has been provided by the TransÉnergie division of HQ. The transmission system includes 2565 buses, 3225 branches, and 290 power plants with a detailed representation of the synchronous machine, its excitation system, automatic voltage regulator, power system stabilizer, turbine, and speed governor. During this thesis, the model was expanded to include 4311 dynamically modeled loads (different types of induction motors, voltages sensitive loads, etc.), to better capture the dynamic response of the real system.

In the long-term the system evolves under the effect of 1111 LTCs, 25 Automatic Shunt Reactor Tripping (ASRT) devices [BTS96], as well as OvereXcitation Limiters (OXL). The resulting model has 35559 differential-algebraic states. The map of the 735-kV grid is shown in Fig. B.6.

1.3.3 PEGASE system

The Pan European Grid Advanced Simulation and State Estimation (PEGASE) project was a four-year R&D collaborative project funded by the Seventh Framework Program (FP7) of the European Union [VLER12]. It was coordinated by Tractebel Engineering and composed of 21

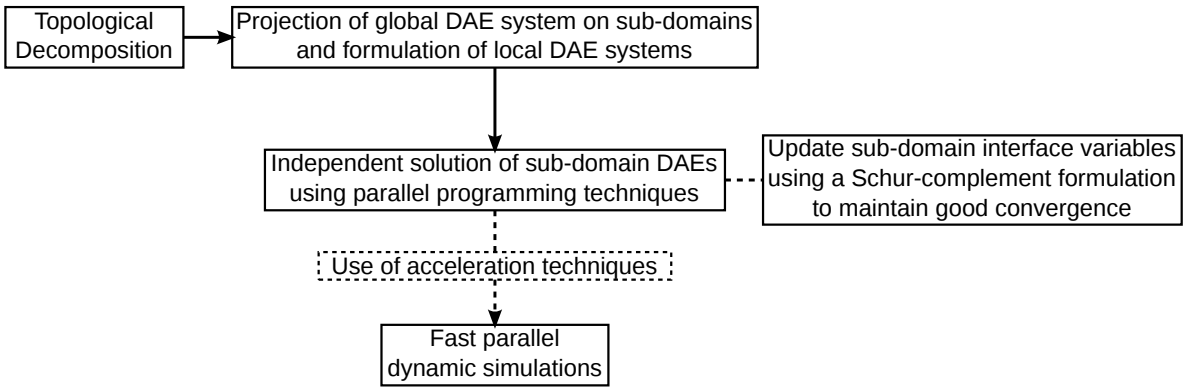


Figure 1.3: Layout of proposed parallel simulation algorithms

partners, including TSOs, expert companies and leading research centers in power system analysis and applied mathematics. The University of Liège was the fourth in terms of budget member of this project.

Within this framework, a test system comprising the continental European synchronous area (see Fig. B.8) was set up by Tractebel Engineering and RTE, the French TSO, starting from network data made available by the European Network of Transmission System Operators for Electricity (ENTSO-E).The dynamic components and their detailed controllers were added in a realistic way. This system includes 15226 buses, 21765 branches, and 3483 synchronous machines represented in detail together with their excitation systems, voltage regulators, power system stabilizers, speed governors and turbines. Additionally, 7211 user-defined models (equivalents of distribution systems, induction motors, impedance and dynamically modeled loads, etc.) and 2945 LTC devices were also included. The resulting model has 146239 differential-algebraic states.

1.4 Thesis objective

In this work, two parallel algorithms belonging to the family of Domain Decomposition Methods (DDMs) are developed to tackle the computational complexity of dynamic simulations of power systems. Both algorithms follow the layout of Fig. 1.3.

The first proposed algorithm is focused on accelerating the dynamic simulation of large interconnected systems. In brief, a non-overlapping, topological-based, decomposition scheme is applied on the model, splitting the network according to the power system components connected to it. This decomposition reveals a star-shaped sub-domain layout and leads to the separation of the DAEs describing the system. Next, the nonlinear DAE system describing each sub-domain is solved independently by algebraizing and using a Newton method with infrequent matrix update and factorization. The interface variables shared between sub-domains are updated using a Schur-complement-based approach.

The most noticeable developments foreseen in power systems involve DNs. Future DNs

are expected to host a big percentage of the renewable energy sources and actively support the transmission grid through smart grid technologies. Despite this, in present-day DSA simulations, it is common to represent the bulk generation and higher voltage (transmission) levels in detail, while the lower voltage (distribution) levels are equivalenced (simplified models are used). On the contrary, when the study concentrates on a DN, the TN is often represented by a Thévenin equivalent. One of the motivations behind this practice has been the lack of computational performance of existing simulation software and the still modest penetration of active DNs.

The second proposed algorithm targets to accelerate dynamic simulations of large combined transmission and distribution systems. It employs a two-level system decomposition. First, the combined system is decomposed on the boundary between the transmission and the DNs revealing a star-shaped sub-domain layout. This first decomposition is reflected to a separation of the DAEs describing the system projecting them onto the sub-domains. Next, a second decomposition scheme is applied within each sub-domain, splitting the sub-domain network from the power system components connected to it, similarly to the first algorithm. This second decomposition further partitions the DAEs describing the system. Finally, the solution of the decomposed DAE systems describing the sub-domains is performed hierarchically with the interface variables being updated using a Schur-complement-based approach at each decomposition level.

The proposed algorithms augment the performance of the simulation in two ways. First, the independent calculations of the sub-systems are parallelized providing computational acceleration. Second, some acceleration techniques are employed, exploiting the locality of the decomposed sub-systems to avoid unnecessary computations and provide numerical acceleration.

The algorithms are first presented with a certain level of abstraction, focusing on the mathematical properties of DDMs, to avoid limiting the implementation to a particular computer architecture. Next, the details concerning their implementation using the shared-memory parallel computing model are presented. Finally, some scenarios are simulated, using the test systems presented above, to show the accuracy and performance of the algorithms.

Both algorithms have been implemented in the academic simulation software RAMSES[2], developed at the University of Liège since 2010, and are used in the context of academic research, as well as in collaborations with industry. The implementation targets common, inexpensive, multi-core machines without the need of expensive dedicated hardware. Modern Fortran and the OpenMP Application Programming Interface (API) are used. The implementation is general, with no hand-crafted optimizations particular to the computer system, operating system, simulated electric power network, or disturbance.

[2] Acronym for “RApid Multithreaded Simulator of Electric power Systems”.

1.5 Thesis outline

Chapter 2 In this chapter, a summary of parallel architectures and techniques is presented. First, the types of parallelism available in modern computers are listed with an effort to categorize them. The theory behind the performance assessment of parallel algorithms and implementations is recalled, followed by some common pitfalls that can hinder their efficiency. Finally, the shared-memory parallel programming model used in this work is detailed.

Chapter 3 This chapter is devoted to DDMs and their applications to power systems. First, the motivation behind the development of DDMs and the essential components that describe these methods are introduced. Next, an effort is made to present previous work on power system dynamic simulation using DDMs and parallel computing.

Chapter 4 In this chapter, the first proposed algorithm is presented. First, the power system decomposition and the formulation of the sub-domain DAEs is described. Next, the solution of the decomposed system and the treatment of the interface variables is detailed. Then, the numerical acceleration techniques are introduced, followed by the description of parallel implementation of the algorithm. Finally, the performance of the algorithm is assessed using the test systems described above.

The mathematical aspects of the algorithm were published in [AFV14] (material accepted for publication in 2012) while, its application to parallel power system dynamic simulations was presented in [AFV13b]. Next, the acceleration techniques were presented in [AFV13a]. A paper focusing on the real-time performance capabilities of the algorithm was presented in [AV14c]. Finally, the complete algorithm was presented at the panel session "Future Trends and Directions in Dynamic Security Assessment" at the 2014 IEEE Power & Energy Society General Meeting in Washington DC [AV14b].

Chapter 5 In this chapter, the second proposed algorithm is presented, featuring a two-level power system decomposition. The hierarchical solution of the decomposed systems and the treatment of the interface variables is detailed, followed by some numerical acceleration techniques used to speedup the procedure. Next, details on the parallel implementation of the algorithm are given. Finally, the performance of the algorithm is assessed using the combined transmission and distribution test systems described above, as well as the Hydro-Québec real system.

Some initial investigations were presented in [AV13], proposing a sequential version of the algorithm with some numerical acceleration techniques. Next, a single-level decomposition algorithm was presented in [AV14a], followed by a journal paper in [AV15]. Finally, a journal paper detailing the proposed two-level DDM has been submitted and is currently under review (see Ref. [16], next page).

Chapter 6 In this chapter, the contribution of this thesis is summarized and some plans for future work are suggested.

Overall, this thesis expands the following material which has been published, submitted, or is under preparation, in various journals and conferences:

Under preparation:

[17] T. Kyriakidis, **P. Aristidou**, D. Sallin, T. Van Cutsem, and M. Kayal. A Linear Algebra Enabled Mixed-Signal Computer for Power System Applications. To be submitted to *ACM Journal on Emerging Technologies in Computing Systems*, 2015

Under review:

[16] **P. Aristidou**, S. Lebeau, and T. Van Cutsem. Fast Power System Dynamic Simulations using a Parallel two-level Decomposition Algorithm. Submitted to *IEEE Transactions on Power Systems (under review)*, 2015.

[15] **P. Aristidou**, G. Valverde, and T. Van Cutsem. Contribution of distribution network control to voltage stability: A case study. Submitted to *IEEE Transactions on Smart Grids (under 2nd review)*, 2015.

[14] F. Plumier, **P. Aristidou**, C. Geuzaine, T. Van Cutsem, "Co-simulation of Electromagnetic Transients and Phasor Models: a Relaxation Approach", Submitted to *IEEE Transactions on Power Delivery (under 2nd review)*, 2015.

Published:

[13] F. Olivier, **P. Aristidou**, D. Ernst, and T. Van Cutsem. Active management of low-voltage networks for mitigating overvoltages due to photovoltaic units. *IEEE Transactions on Smart Grid (in press)*, 2015. Available at: http://hdl.handle.net/2268/172623

[12] **P. Aristidou** and T. Van Cutsem. A parallel processing approach to dynamic simulations of combined transmission and distribution systems. *International Journal of Electrical Power & Energy Systems*, vol. 72, pp. 58–65, 2015.
Available at: http://hdl.handle.net/2268/178765

[11] **P. Aristidou**, S. Lebeau, L. Loud, and T. Van Cutsem. Prospects of a new dynamic simulation software for real-time applications on the Hydro-Québec system. In *Proceedings of 2015 CIGRÉ Canada conference (accepted, final paper due June 15, 2015)*, Winnipeg, September 2015.

[10] **P. Aristidou**, L. Papangelis, X. Guillaud, and T. Van Cutsem. Modular modeling of combined AC and DC systems in dynamic simulations. In *Proceedings of 2015 IEEE PES PowerTech conference* (*in press*), Eindhoven, July 2015.

[9] F. Plumier, **P. Aristidou**, C. Geuzaine, and T. Van Cutsem. A relaxation scheme to combine phasor-mode and electromagnetic transients simulations. In *Proceedings of the 18th Power System Computational Conference (PSCC)*, Wroclaw, August 2014. Available at: http://hdl.handle.net/2268/168630

[8] **P. Aristidou** and T. Van Cutsem. Dynamic Simulations of Combined Transmission and Distribution Systems using Parallel Processing Techniques.In *Proceedings of the 18th Power System Computational Conference (PSCC)*, Wroclaw, August 2014.
Available at: http://orbi.ulg.ac.be/handle/2268/168618

[7] **P. Aristidou** and T. Van Cutsem. Algorithmic and computational advances for fast power system dynamic simulations. In *Proceedings of the 2014 IEEE PES General Meeting*, Washington DC, July 2014. Available at: http://hdl.handle.net/2268/163168

[6] **P. Aristidou**, F. Olivier, D. Ernst, and T. Van Cutsem. Distributed model-free control of photovoltaic units for mitigating overvoltages in low-voltage networks. In *Proceedings of 2014 CIRED workshop*, Rome, June 2014.
Available at: http://hdl.handle.net/2268/165629

[5] **P. Aristidou** and T. Van Cutsem. Parallel computing and localization techniques for faster power system dynamic simulations. In *Proceedings of 2014 CIGRÉ Belgium conference*, Brussels, March 2014. *This paper received the Best Paper student award.*
Available at: http://hdl.handle.net/2268/161322

[4] **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*, 25(10):2561–2570, Oct 2013.
Available at: http://hdl.handle.net/2268/156230

[3] **P. Aristidou** and T. Van Cutsem. Dynamic simulations of combined transmission and distribution systems using decomposition and localization. In *Proceedings of 2013 IEEE PES PowerTech conference*, Grenoble, June 2013. *This paper received the High Quality Paper award.* Available at: http://hdl.handle.net/2268/145092

[2] **P. Aristidou**, D. Fabozzi, and T. Van Cutsem. Exploiting localization for faster power system dynamic simulations. In *Proceedings of 2013 IEEE PES PowerTech conference*, Grenoble, June 2013. Available at: http://hdl.handle.net/2268/145093

[1] **P. Aristidou**, D. Fabozzi, and T. Van Cutsem. A Schur complement method for DAE systems in power system dynamic simulations. In *Domain Decomposition Methods in Science and Engineering XXI , volume 98 of Lecture Notes in Computational Science and Engineering*, Springer International Publishing, 2014 (material accepted for publication in 2012). Available at: http://hdl.handle.net/2268/154312

CHAPTER 2

Think parallel

2.1 The motivation for multi-core processors

Parallelism is a natural feature to humans: we always expect parallel checkout lanes in a supermarket when the number of customers (workload) is sufficiently large. Roads are built with multiple lanes to avoid congestion and few of us would attempt to undertake a huge project (e.g. the construction of a major building) alone.

Serialization is the act of putting some set of operations into a specific order. Decades ago, computer architects started designing computers using serial machine languages to simplify the programming interface. The most important benefit of serial programs is their simplicity: one can read a piece of serial code from top to bottom and understand the temporal order of operations from the structure of the source code. In addition, serial programs are inherently deterministic: they always do the same operations, in the same order, and give the same answer every time you run them with the same inputs. This notion of determinism is useful for debugging, verification, and testing. Deterministic behavior is not guaranteed in parallel programs [MRR12].

Up until the recent past, *frequency scaling* was the dominant reason for improvements in computer performance. The runtime of a program is equal to the number of instructions multiplied by the average time per instruction. Maintaining everything else constant, increasing the clock frequency decreases the average time it takes to execute an instruction. Thus, an increase in frequency decreases runtime for all compute-bound programs [HP02].

However, this is not the case anymore. Modern processors tend to increase the number of cores on the chip rather than the performance of a single core. The reason for the shift to multi-core processors is the increasing difficulty of manufacturers to improve serial performance [Gov10]. Three factors (also known as “walls”) limit the growth in serial performance:

Power wall: unacceptable growth in power usage with clock rate. The power wall results because power consumption (and heat generation) increases nonlinearly as the clock rate increases. Increasing clock rates any further would exceed the power density that

can be dealt with by air cooling, and result in power-inefficient computation [MRR12].

Instructional parallelism wall: limits to available low-level parallelism (see Section 2.2.3). It becomes increasingly more difficulty to find enough parallelism in a single instruction stream to keep a high-performance single-core processor busy.

Memory wall: a growing discrepancy of processor speeds relative to memory speeds [McK04]. This, in effect, pushes for cache sizes to be larger in order to mask the latency of memory. However, it only helps to the extent that memory bandwidth is not the bottleneck in performance.

Nevertheless, the number of transistors that can be put on a single chip is still expected to grow exponentially for many years (Moore's law). So, a simple idea is to use this extra area of silicon to add multiple cores on the same chip, each of lower frequency and power consumption. Thus, the processor now has the potential to do multiple times the amount of work. Moreover, when the processor clock rate falls, the memory wall problems become less noticeable [McK04].

On the one hand, unparallelized applications under-utilize current multi-core processors and leave significant performance on the table. In addition, such serial applications will not improve in performance over time. On the other hand, *efficiently* parallelized applications can fully exploit multi-core processors and should be able to scale automatically to even better performance on future processors. Over time, this will lead to large and decisive differences in performance between sequential and parallel programs.

Knowing the "arsenal" at hand, that is the available parallel computing tools and techniques, permits to detect parallelization possibilities in a computational problem and assess whether the available parallelism is exploitable (is worth the trouble). However, parallel computing is a very active research field with new architectures being invented and new tools implemented every day. It is thus impossible to exhaustively list and detail all of them here.

In this chapter we first try to categorize the main parallel architectures. Then, the theory behind the performance assessment of parallel algorithms and implementations is outlined, followed by some common pitfalls that can hinder their efficiency. Emphasis is given on shared-memory parallel computing techniques and equipment, as these are used later on for this work.

2.2 Types of parallelism

The computations performed by a given program may provide opportunities for parallel execution at different levels: algorithm-level, data-and task-level, instruction-level, or bit-level. Depending on the level considered, tasks of different granularity result.

2.2.1 Algorithm-level parallelism

This is the top-level or coarse-grain parallelism and pertains to a certain level of machine-independent abstraction. In scientific computing problems, algorithm-level parallelism is closely coupled to the mathematical formulation of the problem and the methods used in its solution. It is known that the same scientific problem might be solved with a variety of mathematical tools. However, some of these tools offer higher potential for parallelism than others. Thus, even though a certain method might be the fastest in sequential execution, it is sometimes useful to chose a different method that can be more efficiently parallelized. A well known family of solution methods that offer high potential of parallelism are Domain Decomposition Methods (DDMs) and will be later examined in Chapter 3.

This type of parallelism requires a good knowledge of the underlying problem and its solution mechanics. It is thus impossible for an automatic tool to reformulate an algorithm so as to expose algorithm-level parallelism.

2.2.2 Data-level and task-level parallelism

Data Level Parallelism (DLP) is parallelism in which the *same* operations are being performed on different pieces of data concurrently. Because the operations are on different data, they are known to be independent, which means that dependence checking is not needed. Task Level Parallelism (TLP) focuses on faster execution by dividing calculations onto multiple cores. TLP programs might execute the same or different code on the same or different data.

There is no clear separation between the two and a typical program exhibits both types of parallelism. Moreover, compilers cannot easily find TLP and DLP to exploit in a program, so the programmer must usually perform extra work to specify when this parallelism exists.

2.2.3 Instructional parallelism

Instruction Level Parallelism (ILP) typically refers to how a sequential program run on a single core can be split into micro-instructions. Multiple micro-instructions from subsequent instructions of the same program are then executed concurrently in a pipeline. This type of parallelism is driven by the compiler. For the programmer, this has the advantage that sequential programming languages can lead to a parallel execution of instructions without his/her intervention.

However, the degree of parallelism obtained by ILP is limited, since it is not possible to partition the execution of the instruction into a very large number of steps of equal size. This limit has already been reached for some time for typical processors [RR13] and it becomes increasingly more difficulty to find enough ILP in a single instruction stream to keep a high-performance single-core processor busy.

2.2.4 Bit-level parallelism

At the lowest level is bit-level parallelism. This form of parallel computing is based on increasing processor word size. Increasing the word size reduces the number of instructions the processor must execute in order to perform an operation on variables whose sizes are greater than the length of the word. For example, consider a case where an 8-bit processor must add two 16-bit integers. The processor must first add the 8 lower-order bits from each integer, then add the 8 higher-order bits, requiring two instructions to complete a single operation. A 16-bit processor would be able to complete the operation with a single instruction.

It was a major source of speedup until 32-bit processors became mainstream. While, contemporary desktop processors usually operate on 64-bit data, the main drive behind the 32-bit to 64-bit move was not computational power but rather the need to index more memory.

2.2.5 Types of parallelism used in this thesis

In this dissertation, the algorithm-level, data-level, and task-level parallelisms are used. In Chapters 4 and 5, the parallel algorithms will be first presented using notions from DDMs to outline them in an abstract but concise way. Next, TLP and DLP will be used to describe the parallel algorithm and its implementation.

2.3 Parallel computer hardware

Networks of workstations, massively parallel supercomputers, and multi-processor workstations are just a few of the dozens of different parallel architectures. We will briefly list some of these architectures and try to organize them into a coherent and simple taxonomy. Our primary interest in parallel computer taxonomies is just to help understand the pertinent issues raised by parallel computer hardware and their general applicability to specific algorithms or programming models.

2.3.1 Flynn's taxonomy

By far the most commonly used taxonomy is Flynn's [Fly72]. All computers are characterized according to how many instruction and data streams they have. In Flynn's taxonomy, there are four possibilities:

- **Single Instruction, Single Data (SISD):** This is just a standard non-parallel processor.
- **Single Instruction, Multiple Data (SIMD):** A single operation (task) executes simultaneously on multiple elements of data. SIMD processors are also known as array processors, since they consist of an array of functional units with a shared controller.
- **Multiple Instruction, Multiple Data (MIMD):** Separate instruction streams, each with its own flow of control, operate on separate data. This characterizes the use of multiple

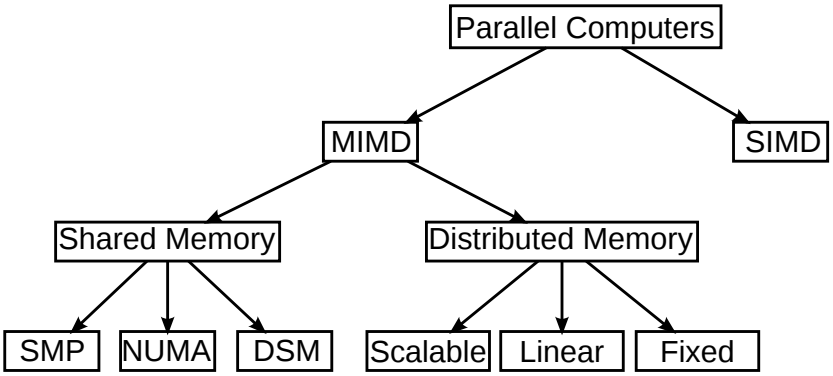


Figure 2.1: Hardware taxonomy according to [MSM04]

cores in a single processor, multiple processors in a single computer, and multiple computers in a cluster. When multiple processors using different architectures are present in the same system, it is called an heterogeneous computer.

- **Multiple Instruction, Single Data (MISD):** This last possible combination is not particularly useful and is not used.

The advantage of Flynn's taxonomy is that it is very well established. Every parallel programmer is familiar with the terms MIMD and SIMD. There are, however, some serious problems, the biggest one being that it provides only four slots to categorize a huge variety of existing systems. This granularity doesn't give us enough ways to separate systems [MSM04].

2.3.2 Further characterization according to memory organization

Unfortunately, computer scientists haven't converged on a single taxonomy for parallel computers. The only consensus is to use Flynn's taxonomy as a starting point. There also seems to be wide agreement that MIMD systems can be further divided based on memory organization [Gur88, MSM04]:

- Shared Memory:
  - Symmetric MultiProcessing (SMP): The memory is physically shared, and all processors access the memory equally at equal speeds. These are sometimes called Uniform Memory Access (UMA).
  - Non-Uniform Memory Access (NUMA): The memory is physically shared, but not distributed in a one to one relation with the processors. Access to different portions of the memory may require significantly different times.
  - Distributed Shared Memory (DSM): The memory is distributed among the processors, but the system gives the illusion that it is shared. It is also called virtual shared memory.

- Distributed Memory:

  - Fixed: The number of connections is fixed as more processors are added (e.g., Ethernet-connected workstations, since the Ethernet interconnection is a single resource that all processors share).
  - Linear: The number of connections grows linearly with the number of nodes (e.g., mesh-connected multicomputers such as the Intel Paragon).
  - Scalable: The number of connections grows as $PlogP$ (where P the number of nodes) or greater (e.g., hypercubes such as the Intel iPSC/860).

The above classification is sketched in Fig. 2.1.

However, even this -more fine- classification is not enough to capture all the different types of parallel architectures available. It is usual that parallel computers fall into more than one category, for example, almost all shared-memory parallel computers also feature SIMD capabilities.

2.4 Selecting a parallel programming model

In computer software, a parallel programming model is a model for writing parallel programs which can be compiled and executed. The selection of this model defines the parallel computer architectures targeted by the application.

The purpose of this work is to accelerate power system dynamic simulations. Such simulations are usually performed using ordinary computers (i.e. desktop workstations or laptops). Most of these computers are already shared-memory multi-core machines (actually, it is almost impossible to purchase a sequential computer nowadays). However, to our knowledge, none of the widely used commercial software offers multithreaded dynamic simulations. This means that they run on one core only, and therefore do not fully utilize the whole power of these parallel computers.

Moreover, the majority of the people performing these simulations do not have access to expensive supercomputers or specialized computer architectures. Thus, while other computer architectures might yield higher performance, we decided early on that our parallel programming model should *at least* target these easily accessible and low-cost shared-memory machines.

Once the type of targeted parallel machines was selected, the available parallel programming models were assessed according to the following criteria:

- Performance: It should be possible to predictably achieve good performance and to scale that performance to larger systems.

- Productivity: It should be easy to program, debug, and maintain. A parallel programming model could be rejected if the outcome source code is obscure, unreadable, and awkward or programming effort is too high in order to achieve the parallelization.

- Portability: Functionality and performance, across operating systems and compilers. It should work on a range of targets, now and in the future. Unfortunately, there is no global consensus on the programming models used. Some models have become *de facto* standards for particular computer architectures but in many cases the selection of a model is strictly a personal choice.

In the following subsections a summary of the parallel programming models considered for this work will be presented.

2.4.1 General purpose computing on graphics processing units

Graphics Processing Units (GPUs) are highly parallel vector processors with local memory for each of the processing cores. The total amount of memory is usually smaller than traditional multi-core CPUs (across different levels of cache) but the number of processing cores is normally larger [LDTY11]. They were originally developed for rendering real-time visual effects in the video gaming industry and are present in all modern computers. Recently, they have become programmable to the point where they can be used as a general purpose programming platform. General purpose programming on the GPU (GPGPU) [FM04, HA11] is currently getting a lot of attention in the scientific community due to the low cost, high availability and computational power. GPUs are really good for fine-grained parallelization and have been used in many linear algebra applications [HKB12, BHSt13].

Power system applications that have GPU-accelerated linear algebra operations have been presented, e.g. power flow studies [GNV07, Gar10, SA10, VMMO11, ADK12, GJY+12, LL14]; state estimation [TDL11, KD13]; transient stability/electromechanical transients simulation [JMD10, JMZD12, BWJ+12, QH13]; electromagnetic transients simulation [DGF12, Cie13, GO13, ZD14]; and optimization, e.g. Optimal Power Flow [RR14].

However, GPUs are not as good in handling the irregular computation patterns (unpredictable branches, looping conditions, irregular memory access patterns, etc.) that most engineering software deal with. Hence, in all cases, the heterogeneous computing concept is used: the CPU assumes main control of the application and the GPU is accelerating the burdensome linear algebra operations. The CPU to GPU data transfer link has relatively high latency introducing a significant bottleneck in the execution of the program [TOG14].

Finally, there is a high effort needed to develop and maintain GPGPU code and low portability as no default standard exists among GPU vendors. Although some programming models have been implemented trying to facilitate programming on such architectures and increase portability of the code (e.g. OpenCL), at the moment of starting this project in 2011

they were still in the early stages without any accumulated experience and documentation. For all the above reasons, this programming model was rejected.

2.4.2 Message passing interface

Message Passing Interface (MPI) has become a *de facto* standard for communication among processes. It is used in the majority of parallel high performance scientific computer programs due to its high performance. In this model, parallel tasks exchange data by passing messages (asynchronous or synchronous) to one another. Although it was designed for execution on distributed memory machines (large networks of workstations or massively parallel supercomputers), this model can also work on shared-memory multi-core machines. It is thus very useful when the programmer expects the application to scale from smaller shared-memory multi-core machines to very large computer clusters in the future.

Among the related power system applications, let us quote: power flow studies [WC76, TAT81, RSI85, SL85, KRF94, AZS96, ZCC96, WHS99, Cha01a, Flu02b, Flu02a, SW03]; voltage stability studies [LT95]; transient stability/electromechanical transients simulation [FP78, OKS90, KPG92, VCB92, CB93, TC95, OO96, WHS99, LBTT90, CI90, LST91, DFK92, LB93, GMLT94, LSS94, DFK96, YXZJ02, CDC02, SXZ05, JMD09, KM09]; optimization, e.g. Optimal Power Flow [KB00, BBMP05]; dynamic security assessment [ABLS97]; state estimation [AT90, KSY+91]; and, electromagnetic transients simulation [VCB92, FKA93].

However, the high communication burden associated to message passing, makes MPI more suitable for coarse-grained parallel algorithms. Algorithms with high rate of data exchange among parallel tasks, as the ones proposed in Chapters 4 and 5, are not likely to be efficient with this parallel programming model. Moreover, in MPI all the message passing is explicit and an experienced programmer is required to implement them correctly, thus the programming effort needed is increased. For these reasons, this programming model was rejected.

2.4.3 Shared-memory models

There exists a plethora of shared-memory parallel programming models. Some well-known examples are OpenCL, Cilk Plus, Intel Threading Building Blocks (TBB), and OpenMP.

OpenCL is a standard organized by Khronos and supported by implementations from multiple vendors. It was primarily designed to allow offload of computation to GPU-like devices, and its memory and task grouping model reflects this. However, OpenCL can also be used for other co-processors as well as shared-memory multi-core CPUs. OpenCL is not intended for mainstream programmers the way TBB, Cilk Plus, or OpenMP are. Lacking high-level programming models for heterogeneous platforms, application programmers often turn to OpenCL. However, over time, higher level models will likely emerge to support mainstream application programmers and OpenCL will be restricted to specialists implementing these higher level models or for detailed performance-oriented libraries [MRR12].

The Cilk (pronounced “silk”) project originated in the mid-1990s at M.I.T. Its successor, Cilk Plus, is integrated with a C/C++ compiler and extends the language with the addition of keywords and array section notation. It uses the fork–join pattern [MRR12] to support irregular parallel programming patterns, parallel loops to support regular parallel programming patterns, and, supports explicit vectorization via array sections, pragma simd, and elemental functions [MRR12].

TBB is a library, not a language extension, and thus can be used with any compiler supporting ISO C++. It relies on templates and generic programming and uses C++ features to implement its “syntax.” TBB requires the use of function objects (also known as functors) to specify blocks of code to run in parallel. Like Cilk Plus, TBB is based on programming in terms of tasks, not threads. This allows it to reduce overhead and to more efficiently manage resources. As with Cilk Plus, TBB implements a common thread pool shared by all tasks and balances load via work-stealing [MRR12].

OpenMP is a standard organized by an independent body called the OpenMP Architecture Review Board. It is based on a set of compiler directives or pragmas in Fortran, C, and C++ combined with an API for thread management. OpenMP is designed to simplify parallel programming for application programmers working in High-Performance Computing (HPC), including the parallelization of existing serial codes. Prior to OpenMP (first released in 1997), computer vendors had distinct directive-based systems. OpenMP standardized common practice established by these directive-based systems and is supported by most compiler vendors including the GNU compilers and other open source compilers [MRR12].

The previous three models are suitable to implement the fork-join parallel pattern associated with “divide-and-conquer” algorithms. Moreover, all three models exhibit high performance, have increased compatibility with existing platforms and compilers, and are widely adopted on shared-memory multi-core computers. However, out of these candidates, the OpenMP API was chosen for our application. The main reason behind this choice is the model’s support of modern Fortran language which is the language of the simulation software RAMSES. More details about the selection of the programming language and the parallel programming model are given in Appendix C.

Some typical power system applications have been investigated using this model, such as: power flow studies [Bos95, DS11, LDTY11, NAA11, Fra12, ASC13]; state estimation [NMT+06, LDTY11, DN12]; transient stability/electromechanical transients simulation [CZBT91, BVL95, HB97, XCCG10, AF12, LDG+12]; and, electromagnetic transients simulation [UBP10, UH11, UD13].

2.5 Performance theory

The primary purpose of parallelization, as discussed in this work, is performance. But what is performance? Usually it is about one of the following:

1. reducing the total time it takes to compute a single result (latency);
2. increasing the rate at which a series of results can be computed (throughput);
3. reducing the power consumption of a computation.

All these valid interpretations of performance can be achieved by parallelization [MRR12].

There is also a distinction between improving performance to reduce costs or to meet a deadline. To reduce costs, one wants to get more done within a fixed machine or power budget and usually is not willing to increase the total amount of computational work. Alternatively, to meet a deadline, it might be necessary to increase the total amount of work if it means the jobs gets done sooner. For instance, in an interactive application (operator training, controller-in-the-loop tests, etc.), it might be required to complete the work fast enough to meet a certain frame rate or response time. In this case, extra work such as redundant or speculative computation [MRR12] might help meet the deadline.

Validation should be given careful thought, in light of the original purpose of the program. Fast computation of wrong answers is pointless, so continuous validation is strongly recommended to avoid wasting time tuning a broken implementation [MRR12]. On the other hand, obtaining results "bit-by-bit identical" to the serial program is sometimes unrealistic. Indeed, the parallel program results, though different, may be as good for the overall purpose as the original serial program, or even better [MRR12].

Frequently, people tend to discuss algorithm performance with all attention focused on the minimization of the total amount of computational work. However, computation may not be the limiting bottleneck. The access to shared memory (or equivalently communication) may constrain performance. Moreover, the potential for scaling performance on a parallel computer is constrained by the algorithm's *span*. The span is the time it takes to perform the longest chain of tasks that *must* be performed sequentially. This chain, also known as the critical path, cannot be speeded up with parallelism no matter how many parallel processors are used. Thus, getting improved performance often requires finding an alternative way to solve a problem that shortens the span [MRR12].

Unsurprisingly, parallel programming is simplest when the parallel tasks are completely independent. In such cases, the span is just the longest task and communication is usually negligible (not zero, because we still have to check that all tasks are done). Parallel programming is much more challenging when tasks are not independent, because that requires communication between tasks, and the span becomes less obvious [MRR12].

2.5.1 Scalability, speedup and efficiency

Three important metrics related to performance and parallelism are speedup, scalability, and efficiency. *Scalability* can be defined as:

$$Scalability_M = \frac{T_1}{T_M} \tag{2.1}$$

where T_1 is the runtime of the program with one worker and T_M is the runtime of the same program, using the same algorithm, with M workers . This index shows how well the parallel implementation scales when the number of available workers is increased.

Optimally, scalability would be linear, i.e. with double the number of processing units the program should execute in half the runtime, and doubling it a second time should again halve the runtime. However, very few parallel algorithms achieve optimal scalability except embarrassingly parallel problems (where there exists no dependency or communication between parallel tasks). Most real-life parallel programs exhibit a near-linear scalability for small numbers of processing units, which flattens out into a constant value for large numbers of workers.

The *speedup* of a parallel implementation expresses the relative saving of execution time that can be obtained by using a parallel execution on M processing units compared to the best sequential implementation. This can be formulated as:

$$Speedup_M = \frac{T_1^*}{T_M} \tag{2.2}$$

where T_1^* is the runtime of the program with one worker using the fastest (or a very fast) sequential algorithm. For power system dynamic simulations, the popular VDHN scheme (see Appendix A) has been suggested as the sequential benchmark [CB93]. Although there is no proof that this is the fastest sequential algorithm, it is employed by many industrial and academic software and its capabilities and performance are well-known.

Moreover, the sequential algorithm needs to be solving exactly the same problem as the parallel with the same accuracy. For this reason, all the algorithms considered in this work (proposed parallel and sequential VDHN) have been implemented in the same software (RAMSES). More precisely, they solve exactly the same model equations, to the same accuracy, using the same algebraization method (namely the second-order BDF), way of handling the discrete events, mathematical libraries (e.g. sparse linear solver), time-stepping strategy, etc. Keeping the aforementioned parameters the same allows for a more rigorous evaluation of the proposed algorithm's performance.

Finally, the *efficiency* of a parallel algorithm is defined as:

$$Efficiency_M = \frac{Scalability_M}{M} = \frac{T_1}{MT_M} \tag{2.3}$$

It is a value, typically between zero and one, estimating how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. Ideally, efficiency should be equal to one, which corresponds to a linear scalability, but many factors can reduce it (as already mentioned) [MRR12]. Algorithms with linear speedup and algorithms running on a single processor have an efficiency of one, while many difficult-to-parallelize algorithms have efficiency such as $\frac{1}{\ln M}$ that approaches zero as the number of processors increases.

In the following subsections, the three main theories for predicting or assessing the performance of parallel algorithm are outlined.

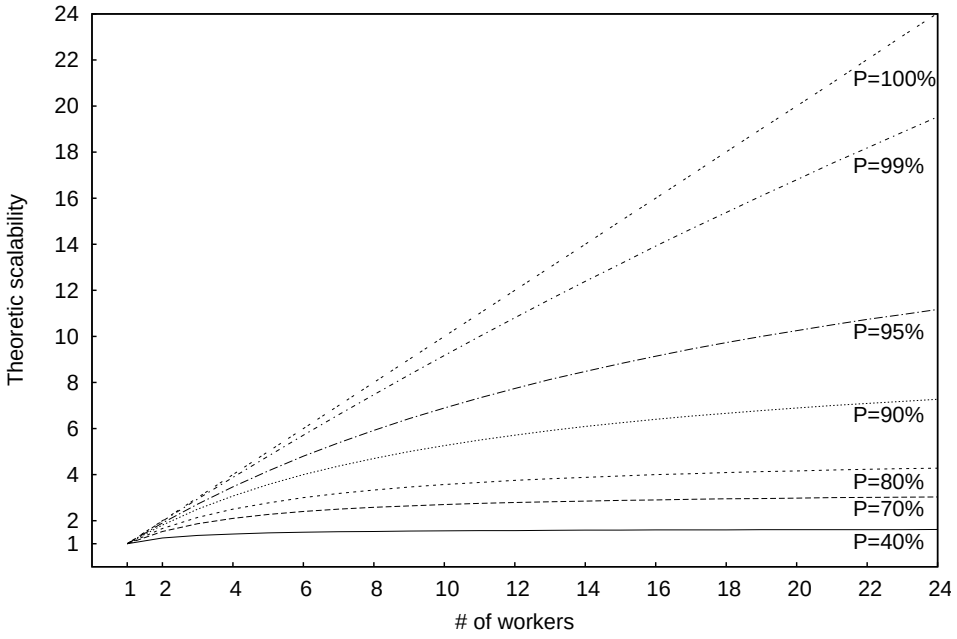


Figure 2.2: Amdahl's formula for scalability

2.5.2 Amdahl's law

The *potential* scalability of an algorithm executed on a parallel computer can be given by a formula originally presented by Gene Amdahl in the 1960s. It states that the portion of the program which cannot be parallelized will limit the overall scalability available from parallelization. A program solving a large mathematical or engineering problem will typically consist of several parallelizable and several non-parallelizable (sequential) parts. The total execution time is described by [Gov10]:

$$T_M = T_S + \frac{T_P}{M} \tag{2.4}$$

where T_S is the time spent in the *serial* part of the code and T_P the time spent in the *parallel* part. Of course, T_P and T_S have to account for 100% of T_1, that is $T_1 = T_P + T_S$. Based on (2.1) and (2.4), scalability is rewritten as:

$$Scalability_M = \frac{T_S + T_P}{T_S + \frac{T_P}{M}} \tag{2.5}$$

Software profiling can give a measurement of T_P and T_S based on the portions of the code scheduled for parallelization. However, if only the semantics ("blueprint") of the algorithm are available, then these values have to be estimated from the mathematical formulation and the *expected* computation times[1].

[1] For the calculations presented in Chapters 4 and 5, the profiling software *Intel VTune Amplifier* was used on a version of the algorithms executed on one core to obtain the values of the parallel and sequential portions.

Using Eq. 2.5, the expected performance of a parallel algorithm can be calculated for a varying number of workers. Figure 2.2 shows these calculations for different values of $P = \frac{T_P}{T_S+T_P}$. It can be seen why linear scalability is extremely difficult to achieve: even with 1% of the computations in sequential execution, the scalability variation differs significantly from linear. Moreover, if the limit $M \to \infty$ is taken on Eq. 2.5, the resulting value shows that the maximum scalability is bounded by the serial time T_S.

Unfortunately, Eq. 2.5 gives an optimistic upper bound of the algorithm's scalability. It assumes that the parallel computations are infinitely divisible and ignores the OverHead Cost (OHC) associated to making the code run in parallel and managing the threads and the communication between them. The communication might be the command for all the threads to start, the exchange of information, or it might represent each thread notifying the main thread that it has completed its work. To compensate for this OHC, Amdahl's law is modified to [Gov10]:

$$T_M = T_S + \frac{T_P}{M} + OHC(M) \tag{2.6}$$

where the $OHC(M)$ is proportional to memory latency for those systems that communicate through memory, or cache latency if all the communicating threads share a common level of cache.

This modified formula suggests that the scalability of an application can be increased either by increasing the percentage of computations in the parallel portion or by reducing the synchronization and communication costs. Also, when OHC is large enough and for small amounts of parallel work (T_P), situations can be encountered where increasing the number of available threads will actually slow down the application. That is, if:

$$T_M - T_{M+1} = \underbrace{\left(\frac{T_P}{M} - \frac{T_P}{M+1}\right)}_{incremental\,gain} + \underbrace{(OHC(M) - OHC(M+1))}_{incremental\,OHC} < 0 \tag{2.7}$$

then adding extra workers to a parallel program can be detrimental to its performance. Unfortunately, there is no way to calculate the exact value of OHC before implementing the parallel code.

2.5.3 Gustafson-Barsis' law

Amdahl's law considers a program as fixed and the parallel computing resources can be varied. However, as computational power increases, applications tend to change to exploit these new features. For example, the demands of power system dynamic simulations have significantly increased compared to 10 or 20 years ago. Simulations today use more detailed power system models. Synchronous machines that were described with the classical model are now much more detailed. Simplified static load models are now replaced by their dynamic counterparts. Distributed energy sources are not anymore just portrayed as negative loads.

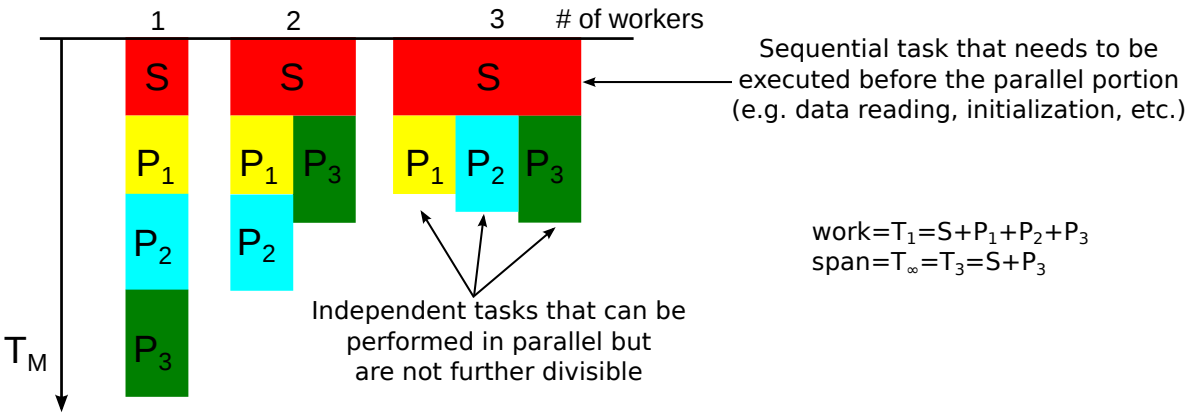


Figure 2.3: Work-span: scalability is limited due to the discretized nature of the parallel tasks

Gustafson-Barsis' law notes that as the problem size grows to take advantage of more powerful computers, the work required for the parallel part of the problem usually grows faster than the serial part. Consequently, the fraction $\frac{T_S}{T_P}$ decreases and scalability improves. This observation should be used as a guideline when deciding the algorithm-level parallelism. If a parallel algorithm is designed correctly, it should take advantage of the future problem scaling to increase its performance automatically.

As mentioned above, the trend in power system simulations is to model in higher detail the electric components. Another target is to include detailed models of DNs and perform combined simulations of TN and DN systems to obtain more consistent results among them [Hed14]. The algorithms proposed in Chapters 4 and 5 were designed to accommodate for this future model expansion in their parallel part, thus leading to higher scalability as the modeling demands increase, with the same parallel computing resources.

Both Amdahl's and Gustafson-Barsis' laws are correct. The difference lies in whether the aim is to make a program run faster with the same workload or run in the same time with a larger workload. History clearly favors programs getting more complex and solving larger problems, so Gustafson's observations fit the historical trend. Nevertheless, Amdahl's law still haunts us when the target is to make an application run faster on the same workload to meet some latency (e.g. real-time simulations) [MRR12].

2.5.4 Work-span model

As mentioned earlier, Amdahl's law makes the assumption that computations in the parallel portion of the algorithm are infinitely divisible. However, this is not true for most applications. In the work-span model, time T_1 is called the *work* of an algorithm. It is the time that the algorithm would take running on one core. Time T_∞ is called the *span* of an algorithm and is the time a parallel algorithm would take on an ideal machine with an infinite number of processors. Alternatively, the span gives the longest chain of tasks that must be executed one after each other [MRR12].

This analysis takes into consideration situations where the parallel computations are discrete, thus increasing the workers beyond a certain number has no effect on scalability. Figure 2.3 shows one such example: increasing the workers to more than *three* has no effect on the execution time as the parallel tasks are not further divisible, thus $T_\infty = T_3 = S + P_3$. So, the serial and the largest indivisible parallel tasks dictate the algorithm's span.

This model gives a more realistic upper limit to the scalability as:

$$Scalability_M = \frac{T_1}{T_M} \leq \frac{work}{span} \tag{2.8}$$

as well as a lower bound, given by Brent's Lemma [Bre74]:

$$T_M \leq \frac{T_1 - T_\infty}{M} + T_\infty \tag{2.9}$$

2.6 Shared-memory computers performance considerations

As mentioned previously, this work targets shared-memory multi-core computers, in which all parallel workers have access to the same shared-memory address space. This machine model makes communication implicit: it happens automatically when one parallel worker writes a value and another one reads it. Thus, shared memory is convenient and facilitates parallel programming.

However, even if the algorithm has high parallelization potential, a careless implementation can easily destroy the performance of the program. Unintended communication, excessive use of synchronization, lack of locality, and load imbalance can lead to increased OHC or, even worse, wrong results.

In this section we will briefly review some of these issues that were investigated for the implementation of the proposed parallel algorithms in RAMSES.

2.6.1 Synchronization

A *race condition* occurs when concurrent tasks perform operations on the same memory location without proper synchronization, and one of the memory operations is a write. Code with a race may sometimes fail unpredictably. Races are not limited to memory locations but can also happen with I/O operations. For example, if two tasks try to print *Hello* at the same time, the output might look like *HeHellllloo*, or might even crash the program [MRR12].

To avoid such problems, the programmer should clearly map the data dependencies of the algorithm parallel tasks. Once this is done, the first priority should be to eliminate as many of these dependencies as possible to *avoid introducing synchronization* that leads to increased OHC. Then, for the remaining dependencies, the appropriate synchronization mechanisms should be used to ensure the correct execution of the program.

OpenMP offers several mechanisms to help the programmer with this task. For example, when a parallel segment is defined, all the data variables accessed by the parallel tasks

can be categorized as *shared* (which means visible and accessible by all threads simultaneously), *private* (which means each thread will have a local copy and use it as a temporary variable), and *firstprivate* (like private except initialized to original value). The clause *default* assigns the default property for all variables.

Other mechanisms available in OpenMP to eliminate data races are the *critical* sections, which define portions of the code to be executed only by one thread at a time, and *atomic*, which ensures that a specific storage location is updated atomically (by only one thread).

Another synchronization technique in OpenMP uses *locks*. Locks are a low-level way to eliminate races by enforcing limits on access to a resource in an environment where there are many threads of execution. A lock is designed to enforce a mutual exclusion concurrency control policy. However, the extensive use of locks can lead to strangling scalability or even deadlocks (when at least two tasks wait for each other and each cannot resume until the other task proceeds). Such low-level mechanisms are not used in RAMSES and thus will not be detailed here.

Finally, it might be useful to use a data race detection program that can identify potential problems in the code. One such program is *Intel Inspector XE*, which has been routinely used to detect data racing problems in RAMSES.

2.6.2 Lack of locality

Locality is another important factor to increase scalability. It refers to two assumptions on future memory accesses, after a worker accesses a location:

- Temporal locality: A worker is likely to access the same location again in the near future.
- Spatial locality: A worker is likely to access nearby locations in the near future.

The locality model asserts that memory accesses close in both time and space are cheaper than those that are far apart. Having good locality in a program can significantly reduce communication OHC [MRR12].

The cost of communication is not uniform and varies depending upon the type of shared-memory architecture and the location of the worker. Small shared-memory machines (e.g. multi-core laptops and office desktops) have UMA architecture; thus each individual processor can access any memory location with the same speed. On the contrary, larger shared-memory machines usually have NUMA architecture, hence some memory may be “closer to” one or more of the processors and accessed faster by them [CJV07].

The main benefit of NUMA computers over UMA is scalability, as it is extremely difficult to scale UMA computers beyond 8-12 cores. At that number of cores, the memory bus is under heavy contention. NUMA is one way of reducing the number of CPUs competing for access to a shared memory bus by having several memory buses and only having a small number of cores on each of those buses.

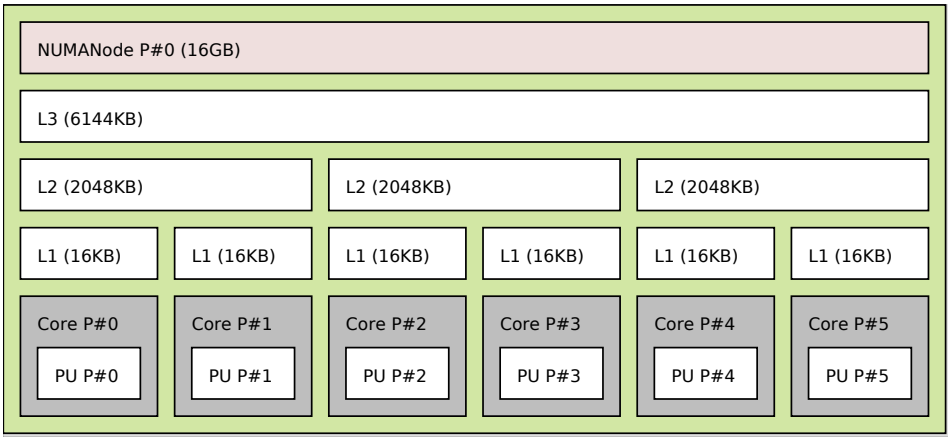


Figure 2.4: cc-Numa architecture used in some of our tests

The cache coherent NUMA (cc-NUMA) node presented in Fig. 2.4 is part of a 48-core NUMA parallel computer, based on 6238 AMD Opteron Interlagos, available at the University of Liège. The computer has *four* identical sockets, each hosting two NUMA nodes with *six* cores (as sketched in Fig. 2.4). So, even though the system physically has four CPU sockets with 12 cores each, there are in fact *eight* NUMA nodes with *six* cores each.

Resources within each node are tightly coupled with a high speed crossbar switch and access to them inside a NUMA node is fast. Moreover, each core has dedicated L1 cache, every two cores have shared L2 cache and the L3 cache is shared between all six cores. These nodes are connected to each other with HyperTransport 3.0 links. The bandwidth is limited to 12GB/s between the two nodes in the same socket and 6GB/s to other nodes. Thus, the cost is minimal for lanes of a vector unit, relatively low for hardware threads on the same core, more for those sharing an on-chip cache memory, and yet higher for those in different sockets [MRR12].

Parallel applications executing on NUMA computers need special consideration to avoid high OHC. First, given the large remote memory access latency, obtaining a program with a high level of data locality is of the utmost importance. Hence, some features of the architecture and Operating System (OS) affect the application's performance (bind threads to particular CPUs, arrange the placement and migration of memory pages, etc.) [CJV07].

Data accessed more frequently by a specific thread should be allocated "close" to that thread. *First Touch* memory allocation policy, which is used by many OS, dictates that the thread initializing an object gets the page associated with that item in the memory local to the processor it is currently executing on. This policy works surprisingly well for programs where the updates to a given data element are typically performed by the same thread throughout the computation. Thus, if the data access pattern is the same throughout the application, the initialization of the data should be done inside a parallel segment using the same pattern so as to have a good data placement in memory. This data initializing procedure is followed in RAMSES.

Some further consideration is needed when large amount of data are read from files to avoid page migration during the initialization. This problem usually affects NUMA machines with low link speed and applications with intensive I/O procedures. In power system dynamic security assessment the data reading is usually done once and then used numerous times to asses several different contingencies on the same system, thus this feature is not critical to their overall performance.

The second challenge on a cc-NUMA platform is the placement of threads onto the computing nodes. If during the execution of the program a thread is migrated from one node to another, all data locality achieved by proper data placement is destroyed. To avoid this we need some method of binding a thread to the processor by which it was executed during the initialization. In the proposed implementation, the OpenMP environment variable *OMP_PROC_BIND* is used to prevent the execution environment from migrating threads. Several other vendor specific solutions are also available, like *kmp_affinity* in Intel OpenMP implementation, *taskset* and *numactl* under Linux, *pbind* under Solaris, *bindprocessor* under IBM AIX, etc.

2.6.3 Load imbalance

One of the most important tasks of parallel programming is to make sure that parallel threads receive equal amounts of work [Cha01b, CJV07]. Imbalanced load sharing among threads will lead to delays, since some of them will be still working while others will have finished and be idle. This is shown in Fig. 2.3, where in the case of three cores, the two of them finish their jobs faster. Load imbalance can be mitigated by further decomposition of the parallel tasks and properly sharing them among the workers [MRR12]. Like packing suitcases, it is easier to spread out many small items evenly than a few big items.

Fortunately, OpenMP offers some mechanisms to facilitate load balancing. For example, when considering the loop-level parallelization (a construct frequently used in engineering problems), the *schedule* clause allows for the proper assignment of loop iterations to threads. However, the best load balancing strategy depends on the computer, the actual data input, and other factors not known at programming time. In the worst case, the best strategy may change during the execution time due to dynamic changes in the behavior of the loop or in the resources available in the system. Even for advanced programmers, selecting the best load balancing strategy is not an easy task and can potentially take a large amount of time.

OpenMP offers three default strategies to assign loop iterations (where each iteration is treated as a task) to threads. With the *static* strategy, the scheduling is predefined and one or more successive iterations are assigned to each thread rotationally *prior* to the parallel execution. This decreases the overhead needed for scheduling but can introduce load imbalance if the workload inside each iteration is not the same. With the *dynamic* strategy, the scheduling is dynamic during the execution. This introduces a high OHC for managing the threads but provides the best possible load balancing. Finally, with the *guided* strategy,

the scheduling is again dynamic but the number of successive iterations assigned to each thread are progressively reduced in size. This way, scheduling OHC is reduced at the beginning of the loop and good load balancing is achieved at the end. Of course, many other, non-standard, scheduling strategies have been proposed in literature [Cha01b].

In general, when deciding the scheduling strategy, the following should be considered:

- optimizing work balancing on the available threads;
- optimizing spatial locality, by assigning tasks that access continuous memory segments to the same thread whenever possible; and,
- optimizing temporal locality, by applying the same schedule in subsequent execution instances of the parallel section to maximize data re-use.

The above performance considerations will be revisited in Chapters 4 and 5, after the parallel algorithms have been defined. The selection of the proper synchronization and scheduling methods will be justified based on the proposed parallel algorithm specifications.

2.7 Description of computers used in this work

In this work, the following computer platforms were used to acquire the simulation results:

1. AMD Opteron Interlagos CPU 6238 @ 2.60GHz, 16KB private L1, 2048KB shared per two cores L2 and 6144KB shared per six cores L3 cache, 64GB RAM, Debian Linux 8 (Fig. 2.4)
2. Intel Core i7 CPU 4510U @ 3.10GHz, 64KB private L1, 512KB private L2 and 4096KB shared L3 cache, 7.7GB RAM, Microsoft Windows 8.1
3. Intel Core i7 CPU 2630QM @ 2.90GHz, 64KB private L1, 256KB private L2 and 6144KB shared L3 cache, 7.7GB RAM, Microsoft Windows 7

Machine 1 is a scientific computing equipment with NUMA architecture, while Machines 2 and 3 are ordinary laptop computers with UMA architecture.

2.8 Summary

Programmers can no longer depend on rising clock rates (frequency scaling) to achieve increasing performance for each new processor generation, due to the power wall. Moreover, they cannot rely on automatic mechanisms to find parallelism in naive serial code, due to the instructional parallelism wall. Thus, to achieve higher performance, they have to write *explicitly* parallel programs [MRR12].

To achieve this, engineers need to revisit the algorithms used for each problem and try to extract the maximum possible algorithm-level parallelism. If necessary, algorithms that

cannot be parallelized should be replaced by more suitable ones. When doing so, the future scaling of the problem size should be considered to ensure that the algorithm's scalability will increase as well. Based on the granularity of the parallel algorithm and the targeted computer architectures, the proper parallel programming model should be selected.

Finally, the programmer should study the communication and memory access patterns to avoid unnecessary OHC that could strangle the scalability of the program.

CHAPTER 3

Domain decomposition methods and applications to power systems

3.1 Introduction

Domain Decomposition Methods (DDMs) refers to a collection of techniques which revolve around the principle of "divide-and-conquer". They were primarily developed for solving large boundary-value problems of Partial Differential Equations (PDEs) by splitting them into smaller problems over sub-domains and iterating between them to coordinate the solution [TW05]. These sub-domains can be overlapping, meaning that some parts of the domain belong to more than one sub-domain, or non-overlapping.

One of the first applications was described by Schwarz in 1869 and consisted of three parts: alternating between two overlapping sub-domains, solving the Dirichlet problem on one sub-domain at each iteration, and taking boundary conditions based on the most recent solution obtained from the other domain [Saa03]. Since then, such methods have been used extensively in problems deriving from physics and structural engineering [Woh01, Saa03, TW05], while the solution over non-overlapping sub-domains through sub-structuring was introduced in the 60's [Prz63].

Next, because of their high efficiency and performance, DDMs became popular in initial value problems coming from the multi-domain mechanical/electrical design and are described by systems of Ordinary Differential Equations (ODEs) or Differential-Algebraic Equations (DAEs). In these problems, the solution domain refers to the states describing the physics and mechanisms of the underlying process and not to the space domain as usually in PDE problems.

In the computer era, DDMs were originally used due to the lack of memory in computing systems: data needed for smaller portions of a problem could fit in the memory while for the whole problem they could not. Moreover, solving smaller problems could prove advantageous as the complexity of several solvers grows more than linearly with the size of

the problem. However, they lost their appeal as larger memory and better scaling solvers became available, only to resurface in the age of parallel computing. These methods are inherently suited for execution on parallel architectures and many parallel implementations have been proposed on multi-core computers, clusters, GPUs, etc.

In this chapter we overview some of the features that define any given DDM. Then, we focus on DDMs proposed for dynamic simulations of power systems.

3.2 DDM characteristics

Any DDM needs to answer three questions: how is the problem domain partitioned into sub-domains, how are the sub-domain problems formulated and solved, and how are the sub-domain interface variables processed. The answers to these questions provide the "blueprint" of the method. These DDM features are described next, with focus on the peculiarities appearing in DAE problems.

3.2.1 Sub-domain partitioning

The first step in designing a DDM is to identify the preferred characteristics of sub-domains. This includes choosing the number of sub-domains, the type of partitioning, and the level of overlap. Each of these choices depend on a variety of factors such as size and type of the domain, the number of parallel processors, communication cost, and the system's underlying dynamics.

The main target when partitioning the domain is to minimize the interfaces between the sub-domains. This will allow for lower communication requirements and a simpler handling of interface variables. In PDE problems, where DDMs have been mostly applied, the decomposition is usually based on the geometrical data and the order of the discretization scheme used [Saa03, TW05]. Conversely, in DAE/ODE problems (such as the one under consideration in this work), no *a priori* knowledge of the coupling variables is available since there are no regular data dependencies (such as those defined by geometric structures). In several cases, the so-called *dependency matrix* ($\boldsymbol{D}$) can be used. For a system with N equations and N unknown variables, $\boldsymbol{D}$ is an NxN matrix with $\boldsymbol{D}(i,j) = 1$, if the i-th equation involves the j-th variable, and $\boldsymbol{D}(i,j) = 0$ otherwise. However, each system model can be composed of several sub-models which are sometimes hidden, too complex, or used as black boxes. Hence, an automatic calculation of $\boldsymbol{D}$ is not trivial to implement [GTD08].

Moreover, in so far as the sub-problems obtained from a given partitioning are solved in parallel, there are some restrictions regarding the type of partitioning needed. For example, the DAE model of a power system component (e.g. synchronous machine, wind turbine, etc.) has a dense dependency matrix. Thus, the component should *not* be split between sub-domains as it will create a very strong connection between different sub-domains, increase the communication cost, and decrease the efficiency of the DDM.

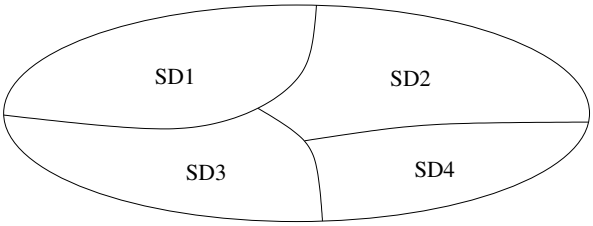


Figure 3.1: Example of decomposed system

Overall, decomposition schemes for DAE/ODE systems have to rely on problem specific techniques which require good knowledge of the underlying system, the models composing it, and the interactions between them. A bad selection of the system partitioning can lead to the DDM solution not converging or converging very slowly.

3.2.2 Problem solution over sub-domains

The second feature that characterizes a DDM is the way of solving the sub-problems formulated after the partitioning. Given the decomposed system sketched in Fig. 3.1, the i-th sub-domain SD_i ($i = 1,..,4$) is described by a local problem (sub-problem) involving three kind of variables:

1. interior variables which are coupled through local equations ($\boldsymbol{x}_i^{int}$) only;
2. local interface variables which are coupled through both local and non-local (external to the sub-domain) equations ($\boldsymbol{x}_i^{ext}$); and,
3. external interface variables that belong to other sub-domains and are coupled through local equations ($\boldsymbol{x}_j^{ext}$).

In general, standard methods for treating PDE, ODE, or DAE problems are employed to solve each sub-system. For example, to solve DAE systems coming from dynamic simulations of power systems, the techniques presented in Section 1.2 can be used.

Next, it is decided whether the sub-problems will be solved approximately or exactly before exchanging information (updating the interface values) with other sub-domains. This decision always leads to a compromise between numerical convergence speed and data exchange rate. If the interface values are frequently exchanged, global convergence is usually better since the sub-domain solution methods always use recent values of interface variables. However, this leads to higher data exchange rate. If the interface values are infrequently exchanged, there is lower data exchange rate. Nevertheless, the global convergence might degrade since the sub-domain solution methods use older interface values.

Usually, when the sub-domains are weakly connected, solving accurately and avoiding to update often the interface values is better. This kind of partitioning, though, might be very difficult, or even impossible, to obtain. So, updating the interface values more often might be required to improve the convergence rate.

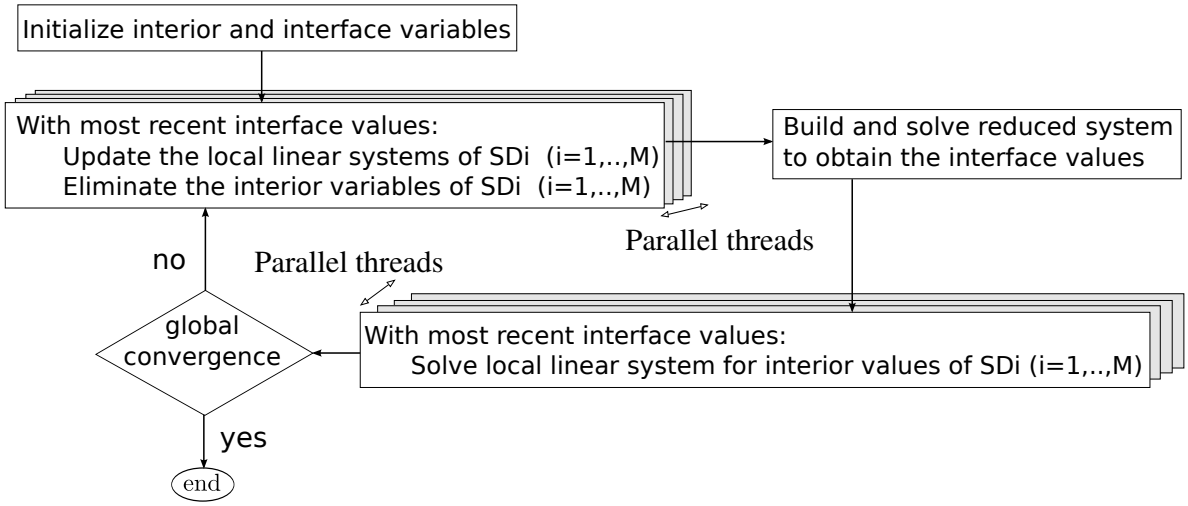


Figure 3.2: Schur-complement-based DDM template

3.2.3 Sub-domain interface variables processing

No matter the partitioning chosen, the sub-domains need to communicate information between them through the interface variables (unless totally disconnected sub-domain partitioning is possible). The values of these variables can be either successively updated through iterations during the solution procedure or computed directly. In general there are two main families of methods for treating the interface values of a decomposed system: Schur-com*plement* and *Schwartz alternating methods*.

3.2.3.1 Direct solution with Schur-complement method

When applying the Schur-complement method, also called *iterative* sub-*structuring*, the original physical domain is split into *non-overlapping* sub-domains. Next, an iterative method (e.g. Newton's method) is used to solve the sub-problems by formulating a, corresponding to each sub-domain, local linear system. The Schur-complement technique is a procedure to eliminate the interior variables in each sub-domain and derive a global, reduced in size, linear system involving only the interface variables. This reduced system can be solved to obtain the interface variables [Saa03].

Once the interface values are known, the sub-problems are decoupled and the remaining, interior to each sub-domain, unknowns can be computed independently. In many cases, building and solving the reduced system involves high computational cost. Many methods have been proposed to speed up the procedure, such as approximately solving the reduced system [SS99], assembling the matrix in parallel using the "local" Schur-complements [Saa03], using Krylov solvers [GTD08], or exploiting the structure of the decomposition to simplify the problem.

In general, the Schur-complement method can be described by the flowchart of Fig. 3.2 (where M is the number of sub-domains). This method can be easily parallelized as sketched

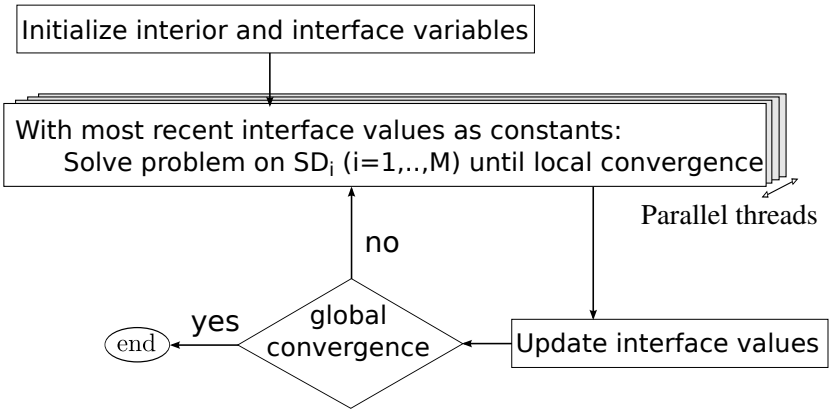


Figure 3.3: Schwarz Alternating procedure DDM template

in the same figure. Operations without data dependencies, such as the formulation and update of the local linear systems, the elimination of the interior variables and the solution of the local systems can be performed concurrently. Nevertheless, the solution of the reduced system introduces an unavoidable sequential bottleneck in the algorithm, which can hinder its scalability (see Section 2.5.2).

3.2.3.2 Schwarz Alternating procedures

As noted in Section 3.1, among the simplest and oldest techniques are the Schwarz Alternating procedures. In general, these methods can be described by the flowchart of Fig. 3.3.

These methods work by freezing the external interface variables during the solution of each sub-domain, thereby making the sub-problems totally decoupled. Hence, no reduced system has to be formulated and solved. This formulation is more attractive for parallel implementations since the solutions of the sub-domains can be performed in parallel and information exchange happens only when updating the interface values and checking global convergence. Other variants of this method can be found in the literature depending on how often and in which order the interface variables are updated, for instance the Additive or Multiplicative Schwartz procedures [Saa03].

Although these algorithms allow for a more coarse-grained[1] parallelization and have a smaller data exchange rate than the Schur-complement ones, the speed of global convergence of the system can suffer if certain aspects are not taken into consideration [BK99, PLGMH11]. For example, even if the method converges for any chosen partitioning, the choice of the partitioning has a great influence on the rate of the convergence: tightly coupled sub-domains can initiate many iterations as the interface values change strongly and,

[1] Parallel applications are often classified according to how often their subtasks need to synchronize or communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if they do not communicate many times per second, and it is embarrassingly parallel if they rarely or never have to communicate. Embarrassingly parallel applications are considered the easiest to parallelize and in this category belongs the simulation of several *independent* contingencies during a dynamic security assessment.

eventually, slow down the procedure. Choosing carefully the partitioning scheme and the initial values, using preconditioners and overlapping sub-domains can remedy some of these problems. Other, problem specific, considerations involve choosing the right interface variable transmission conditions and discretization method [Woh01, TW05, CRTT11].

3.3 Existing approaches in power system dynamic simulations

DDMs have been contemplated in power system dynamic simulations since the early 1960's. In Fig. 3.4 we attempt to classify these methods according to their decomposition granularity and the way of handling the interface variables.

A clear distinction between them is not always possible. For example, a coarse-grain scheme can be always reformulated as a fine-grain scheme, if the decomposition is exploited only for the solution of the sparse linear system involved in the simulation. This is shown by the dotted lines in Fig. 3.4, where the coarse-grain methods can lead back to the fine-grain ones. However, a coarse-grain scheme allows to parallelize more than just the linear system solution: other steps such as discretization and linearization of the DAE systems, handling of discrete events, convergence check, etc. can be performed concurrently as well, thus increasing the granularity.

The remaining of this section is devoted to commenting on the classification in Fig. 3.4.

3.3.1 Partitioning

The choice of the decomposition plays important role in the speed of convergence, the load balancing among parallel tasks and the overall performance of the DDM. As discussed in Section 3.2, the automatic partitioning of systems described by DAEs is not trivial. Several partitioning schemes have been proposed in power system dynamic simulations.

Some partitioning methods are based on coherency analysis [Pod78, HR88, KRF92, YRA93, JMD09] and take into consideration the dynamic behavior of the system to ensure that the resulting sub-domains are "weakly" coupled. However, this type of partitioning is strongly dependent on the post-fault power system topology and the initial operating point. It is also computationally demanding to update the coherent groups after major system changes.

Other algorithms try to partition the power system in order for the admittance matrix of the resulting sub-networks to be in block border diagonal form (see Fig. 3.5). Such methods are factorization path tree partitioning [Cha95], tearing using simulated annealing [IS90], clustering by contour tableau [SVCC77], and seed nodes aggregating [VFK92].

Finally, some methods [MQ92, CRTT11, AF12, ASC13] build a graph representation of the system and try to partition it with the objective of minimizing the edge cuts while having balanced sub-domains.

Figure 3.4: Overview of DDMs in power system dynamic simulations

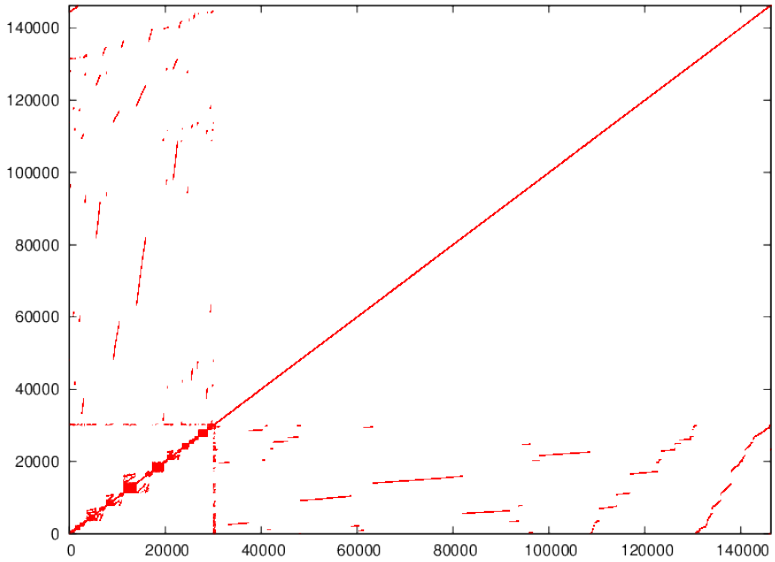


Figure 3.5: Block bordered diagonal form matrix for power system dynamic simulation

3.3.2 Fine-grained methods

With the huge developments in parallel computing in the 80's, researchers started investigating the use of parallel resources to accelerate the solution of sparse linear systems, present in many engineering problems. Several parallel solvers have been developed since then, either of general purpose or customized addressing a specific matrix structure.

In power system dynamic simulations, decomposition methods have been used to to identify and exploit the particular structure of sparse matrices and efficiently parallelize the solution procedure. Some methods, like parallel VDHN [CB93], Newton W-matrix [YXZJ02] and parallel LU [Cha01a, CDC02], divide the independent vector and matrix operations involved in the linear system solution over the available computing units. These methods solve the *exact* linear system (e.g. Eq. 1.15) but in parallel. Other methods, like parallel successive over relaxed Newton [CZBT91] and Maclaurin-Newton [CB93], use an approximate (*relaxed*) Jacobian matrix with more convenient structure for parallelization. Afterwards, several methods were proposed inspired by different hardware platforms and parallel computing memory models [TC95, SXZ05, FX06, JMD10].

While the fine-grained parallelization methods provide some speedup, they don't exploit the full parallelization potential of power system dynamic simulations. First, only parallelization opportunities deriving from linear algebra operations are exploited; leaving the procedures performing the discretization and algebraization of equations, the treatment of discrete events, the convergence check, etc. in sequential execution. Furthermore, as these methods handle the sparse matrix directly, it is really hard to perform partial system updates or exploit the localized response of power systems to certain events (such acceleration techniques will be presented in Chapters 4 and 5). Finally, due to the fine-grain partitioning of tasks there

is a high rate of communication among the different parallel workers, leading to increased overhead costs. For these reasons, coarse-grained DDMs were preferred in our work.

3.3.3 Coarse-grained methods

In coarse-grain methods the decomposition is not projected to the linear system but to the power system DAEs of Eq. 1.1. This leads to the formulation of the DAE sub-systems describing the problem in the resulting sub-domains. Then, the solutions of the decomposed sub-systems are obtained using a Schwarz or Schur-complement method for treating the interface variables, as described in Section 3.2.

Historically, Schwarz methods were preferred for almost all coarse-grain DDMs for power system dynamic simulations. These schemes are easier to formulate and solve, they avoid the sequentiality of Schur-complement methods, and their low rate of communication leads to decreased overhead costs. However, their performance is very sensitive to the particular partitioning and initial operating point.

The first to envisage an application to power systems was probably Kron with the *diakoptics* method [Kro63, Hap74, Ait87]. At the time it was first proposed (1960's), parallel computing was not an option and the target was to address memory issues, but, this method provided the ignition for many of the parallel methods to follow.

First, the network (domain) is teared (decomposed) to create sub-networks (sub-domains). The partitioning is performed using a graph representation of the network and cutting over connecting lines (edges of the graph). It consists of two types [Hap74]: i) the resulting sub-domains form a radial network when the cut lines are removed, and, ii) the resulting sub-domains are no longer connected when the cut lines are removed. Then, the resulting sub-problems are solved independently as if they were completely decoupled. Finally, their solutions are combined to yield the solution of the original problem.

The diakoptics belongs to the family of *parallel-in-space* methods: the formulation of the problem at a single time instant is decomposed and solved. Moreover, the interface variables are updated through a Schwarz approach. The performance of this method is severely decreased as the network to be partitioned is more meshed.

Another family of methods is the *parallel-in-time* [Alv79, LBTT90, LST91, LSS94, ILM98]. These introduced the idea of exploiting parallelization in time and have their origins in similar techniques proposed for the solution of ODE problems [Nie64]. So, despite the sequential character of the initial value problem which stems from the discretization of differential equations, these techniques propose the solution of multiple consecutive time instants in parallel. Thus, the DAE systems describing the problem at each instant are solved concurrently. The interface variables exchanged refer to the transfer of the values needed by the discretization scheme from one time instant to the next, as shown in Fig. 3.6. For example, with a two-step discretization scheme information from the two previous time instants would be always needed.

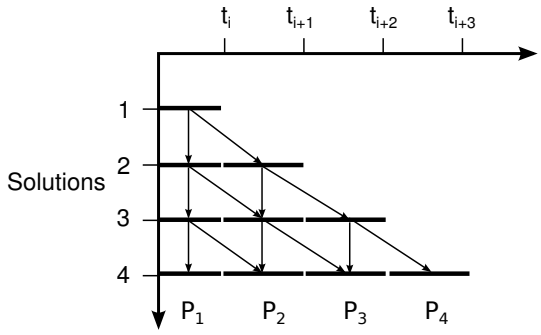


Figure 3.6: Parallel-in-time techniques: propagation of information on four processors

Later on, the *Waveform Relaxation* (WR) [ISCP87, CIW89, CI90] method proposed to decompose the system in space and solve each sub-domain for several time instances before exchanging the interface values. Hence, these values consist of waveforms from neighboring sub-systems (i.e. a sequence of interface values over a number of consecutive time instants). After each solution, waveforms are exchanged between neighboring sub-systems, and this process is repeated until global convergence. Thus, these schemes follow a Schwarz approach to treat the interface variables.

Consider the DAE initial value problem of (1.1) solved over the time window $[t_0,\, t_0 + T_W]$:

$$\begin{aligned} \boldsymbol{\Gamma}\dot{\boldsymbol{x}} &= \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{V}) \\ \boldsymbol{0} &= \boldsymbol{\Psi}(\boldsymbol{x}, \boldsymbol{V}) \\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0,\, \boldsymbol{V}(t_0) &= \boldsymbol{V}_0 \end{aligned}$$

Decomposing the power system into M sub-domains yields M sets of equations, each one describing the problem inside its partition:

$$\begin{aligned} \boldsymbol{\Gamma}_1\dot{\boldsymbol{x}}_1^{k+1} &= \boldsymbol{\Phi}_1(\boldsymbol{x}_1^{k+1}, \boldsymbol{x}_2^k, ..., \boldsymbol{x}_r^k, \boldsymbol{V}_1^{k+1}, \boldsymbol{V}_2^k, ..., \boldsymbol{V}_M^k) \\ 0 &= \boldsymbol{\Psi}_1(\boldsymbol{x}_1^{k+1}, \boldsymbol{x}_2^k, ..., \boldsymbol{x}_r^k, \boldsymbol{V}_1^{k+1}, \boldsymbol{V}_2^k, ..., \boldsymbol{V}_M^k) \\ \boldsymbol{\Gamma}_2\dot{\boldsymbol{x}}_2^{k+1} &= \boldsymbol{\Phi}_2(\boldsymbol{x}_1^k, \boldsymbol{x}_2^{k+1}, ..., \boldsymbol{x}_r^k, \boldsymbol{V}_1^k, \boldsymbol{V}_2^{k+1}, ..., \boldsymbol{V}_M^k) \\ 0 &= \boldsymbol{\Psi}_2(\boldsymbol{x}_1^k, \boldsymbol{x}_2^{k+1}, ..., \boldsymbol{x}_r^k, \boldsymbol{V}_1^k, \boldsymbol{V}_2^{k+1}, ..., \boldsymbol{V}_M^k) \\ &\vdots \\ \boldsymbol{\Gamma}_M\dot{\boldsymbol{x}}_M^{k+1} &= \boldsymbol{\Phi}_M(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k, ..., \boldsymbol{x}_r^{k+1}, \boldsymbol{V}_1^k, \boldsymbol{V}_2^k, ..., \boldsymbol{V}_M^{k+1}) \\ 0 &= \boldsymbol{\Psi}_M(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k, ..., \boldsymbol{x}_r^{k+1}, \boldsymbol{V}_1^k, \boldsymbol{V}_2^k, ..., \boldsymbol{V}_M^{k+1}) \end{aligned} \tag{3.1}$$

The variables with superscript $k+1$ in the sub-sets are unknowns and have to be calculated over the window $[t_0,\, t_0 + T_W]$, while the variables with superscript k are known from the previous WR iteration and make up the interface variables exchanged between the systems at the end of each iteration.

For the solution of the system (3.1) two algorithms were proposed [CI90]: Gauss-Jacobi WR Algorithm and Gauss-Seidel WR Algorithm (see Fig. 3.7). The former uses boundary

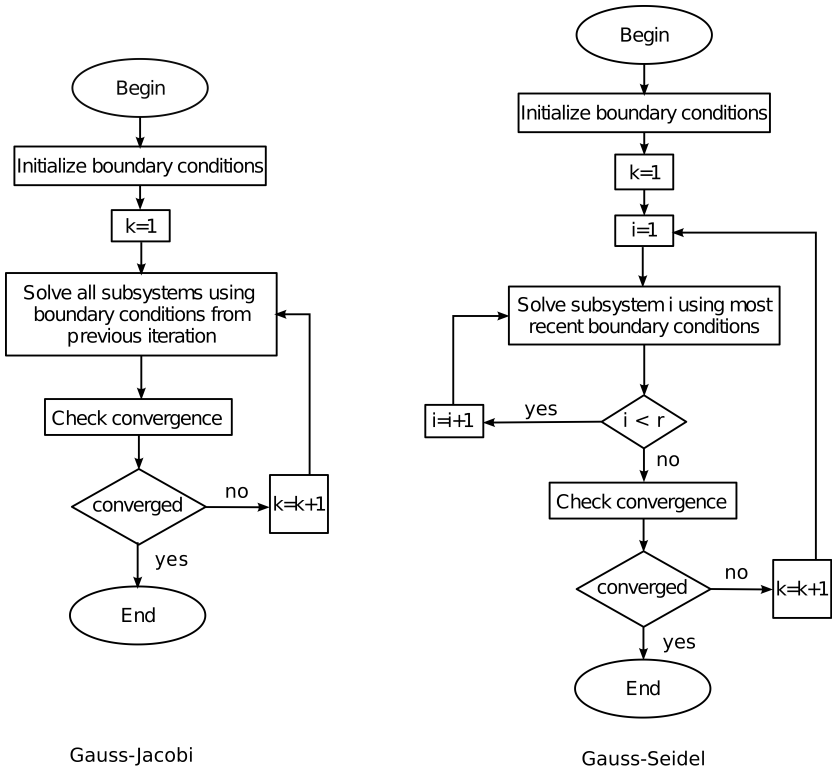


Figure 3.7: WR algorithms over the time interval $[t_0,\, t_0 + T_W]$

conditions only from the previous iteration while the latter uses the most recent boundary conditions available. This way, the first one is inherently suitable for parallel implementation, while the latter for sequential. The convergence of these algorithms strongly depends on the partitioning and on the window time interval T_W [PLGMH11]. Their properties have been thoroughly investigated in [CIW89, BDP96, BK99, GHN99, JW01, EKM08, CRTT11] and some techniques to accelerate this method in [PLGMH11].

If the time interval $[t_0,\, t_0 + T_W]$ of the WR is restricted to *only one time step*, then the *Instantaneous Relaxation* (IR) method [JMD09] is formulated. However, as a Schwartz method, the convergence properties of IR heavily depend on the proper selection of the partition. Furthermore, in the proposed IR method, each sub-system is solved until convergence but the global convergence of the system is not checked before proceeding to the next time instant. This is made possible due to the small time steps used for the simulation ($\sim$1 ms), but could lead to inaccuracies. An extension of this method was proposed in [JMZD12], combining both coarse-grained and fine-grained parallelization in a nested way to increase performance.

Finally, methods such as *parallel-in-time-and-space* [LBTC90], *Parareal-Waveform Relaxation* [CM11], and *Two-Stage Parallel Waveform Relaxation* [LJ15] propose the decomposition of the system in space and in time, to exploit a higher level of parallelization and further speed-up the simulations.

The DDMs proposed in this thesis are coarse-grain and, unlike most of the algorithms proposed until now, make use of the Schur-complement approach to treat the interface vari-

ables. This approach makes the algorithm performance less dependent on the selected partition and allows to achieve high convergence rate. These will be detailed in Chapters 4 and 5.

3.4 Summary

DDMs have been extensively used in engineering for the solution of complex problems described by PDEs, ODEs, and DAEs. The various proposed methods might differ in the type of domain partitioning, the procedure to solve the sub-problems formulated over the partitions, and the way of handling the interface variables. However, the common factor is that they try to exploit parallel computational resources and accelerate the simulation procedure.

A classification of DDMs applied for power system dynamic simulations has been proposed in this chapter followed by a brief description of these methods. Even though several parallel DDMs have been proposed over the years, to our knowledge, industry-grade software do not employ such methods. We believe that the main reason for this absence is the lack of robustness in the solvers and the need for the user to go through a highly complex procedure of parameter and partition selection. Most of the proposed DDMs heavily depend on the selected partition scheme and the latter can easily lead to convergence problems or inaccuracy.

CHAPTER 4

Parallel Schur-complement-based decomposition method

4.1 Introduction

In this chapter, a parallel, Schur-complement-based Domain Decomposition Method (DDM) is presented for the dynamic simulation of power systems. In brief, a non-overlapping, topological-based, decomposition scheme is applied, splitting the system network from the components connected to it. This decomposition reveals a star-shaped sub-domain layout and leads to the separation of the Differential-Algebraic Equations (DAEs) describing the system. Then, the nonlinear DAE system describing each sub-domain is solved independently by algebraizing and using a Newton method with infrequent matrix update and factorization. The interface variables shared between sub-domains are updated using a Schur-complement approach, at each iteration.

The proposed algorithm augments the performance of the simulation in two ways. First, the independent calculations of the sub-systems are parallelized providing computational acceleration. Second, three acceleration techniques are employed, exploiting the locality of the decomposed sub-systems to avoid unnecessary computations and provide numerical acceleration. The algorithm is first presented with a certain level of abstraction, focusing on its mathematical formulation. Next, the details concerning its implementation using the shared-memory parallel computing model are presented. Finally, some results are presented from the test systems of Section 1.3 to assess the accuracy and performance of the algorithm.

It has to be noted that the solution algorithm and the localization techniques proposed in this chapter have their origins in [FCHV13]. However, this work extends and enhances these ideas. First, the solution algorithm is presented using DDM semantics; thus, allowing to better classify and analyze the algorithm. Second, the *sequential* procedure presented in [FCHV13] is parallelized to fully exploit computational resources. Third, the criteria used in the "*latency*" localization technique (to be presented in Section 4.6.3) are revised to make it

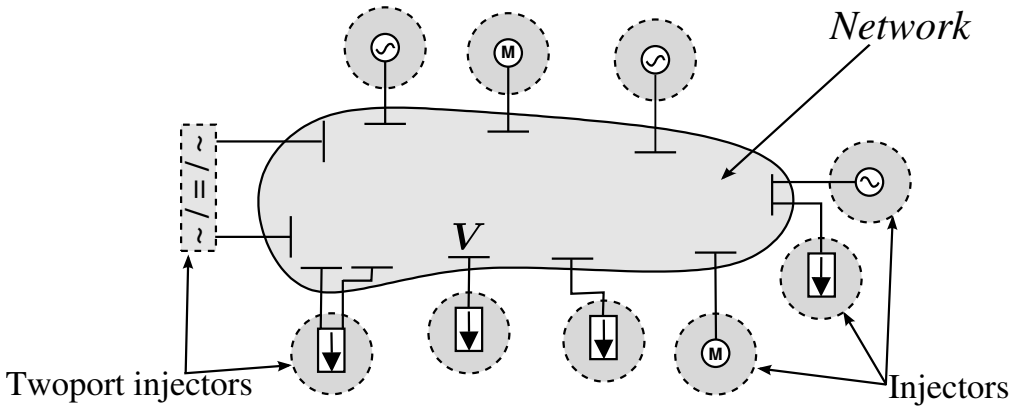


Figure 4.1: Decomposed power system

more accurate and robust. Finally, the theoretical basis is set to analyze the convergence of the proposed algorithm and how it is affected by the localization techniques.

4.2 Power system decomposition

The first step in applying a DDM is to select the domain partitioning. For the proposed algorithm, the electric network is first separated to create one sub-domain by itself. Then, each component connected to the network is separated to form the remaining sub-domains. The components considered in this study refer to devices that either produce or consume power in normal operating conditions and can be attached to a single bus (e.g. synchronous machines, motors, wind-turbines, etc.) or on two buses (e.g. HVDC lines, AC/DC converters, FACTS, etc.). Hereon, the former components will be simply called *injectors* and the latter *twoports*. In the following material, twoports will be explicitly referred *only* when their treatment is different from the injectors; otherwise, only the injectors will be referred.

Components with three or more connecting buses have not been considered in this work; however, such components can be treated as a combination of twoports. For instance, a component connected to three buses can be treated as three twoports. Nevertheless, such components are rarely used in phasor mode power system dynamic simulations and are not present in the test systems considered in Section 1.3.

The proposed decomposition can be visualized in Fig. 4.1. The scheme chosen reveals a star-shaped, non-overlapping, partition layout. At the center of the star, the network sub-domain has interfaces with many smaller sub-domains; while, the latter interface only with the network sub-domain and not between them. As it will be seen later on, this type of partitioning facilitates and simplifies the use of the Schur-complement approach to treat the interface variables. Based on this partitioning, the problem described by Eq. 1.1 is decomposed as follows.

The network sub-domain is described by the algebraic equations:

$$\begin{aligned} \mathbf{0} &= \boldsymbol{\Psi}(\boldsymbol{x}^{ext}, \boldsymbol{V}) \\ \boldsymbol{x}^{ext}(t_0) &= \boldsymbol{x}_0^{ext},\ \boldsymbol{V}(t_0) = \boldsymbol{V}_0 \end{aligned} \tag{4.1}$$

while the sub-problem of each injector can be described by a DAE IVP ($i = 1, ..., N$):

$$\begin{aligned} \boldsymbol{\Gamma}_i \dot{\boldsymbol{x}}_i &= \boldsymbol{\Phi}_i(\boldsymbol{x}_i, \boldsymbol{V}^{ext}) \\ \boldsymbol{x}_i(t_0) &= \boldsymbol{x}_{i0},\ \boldsymbol{V}^{ext}(t_0) = \boldsymbol{V}_0^{ext} \end{aligned} \tag{4.2}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{\Gamma}_i$ are the projections of $\boldsymbol{x}$ and $\boldsymbol{\Gamma}$, defined in (1.1), on the i-th sub-domain. Furthermore, the variables of each injector $\boldsymbol{x}_i$ are separated into interior $\boldsymbol{x}_i^{int}$ and local interface $\boldsymbol{x}_i^{ext}$ variables and the network sub-domain variables $\boldsymbol{V}$ are separated into interior $\boldsymbol{V}^{int}$ and local interface $\boldsymbol{V}^{ext}$ variables. The external interface variables of the network are the rectangular components of the injector currents (see Section 1.2.6), while, the external interface variables of each injector are the coordinate voltage components of the connection bus.

An important benefit of this decomposition is the modeling modularity added to the simulation software and the separation of the injector modeling procedure from the solver. The predefined, standardized interface between the network and the injectors permits for the addition or modification of an injector in an easy way.

4.3 Sub-system solution

For the solution of each sub-system, the techniques detailed in Section 1.2 are used. First, for the solution of the network sub-system, its algebraic equations are formulated as described in (1.3):

$$\mathbf{0} = \boldsymbol{D}\boldsymbol{V} - \boldsymbol{I} = \boldsymbol{D}\boldsymbol{V} - \boldsymbol{x}^{ext} = \boldsymbol{D}\boldsymbol{V} - \sum_{i=1}^{N} \boldsymbol{C}_i \boldsymbol{x}_i \triangleq \boldsymbol{g}(\boldsymbol{x}^{ext}, \boldsymbol{V}) \tag{4.3}$$

where the external interface variables are the rectangular components of the injector currents ($\boldsymbol{I}_i$) and $\boldsymbol{C}_i$ is a trivial matrix with zeros and ones whose purpose is to extract the interface variables from $\boldsymbol{x}_i$.

Then, the injector DAE sub-systems (4.2) are algebraized using a differentiation formula (in this work the second-order BDF) to get the corresponding non-linear algebraized systems:

$$\mathbf{0} = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{V}^{ext}),\ \ i = 1, \ldots, N. \tag{4.4}$$

Next, at each discrete time instant t_n, each of the $N + 1$ sub-systems is solved using a Newton method. Thus, at the k-th Newton iteration the following linear systems are solved:

$$\boldsymbol{D}\,\Delta\boldsymbol{V}^k - \sum_{i=1}^{N} \boldsymbol{C}_i \Delta \boldsymbol{x}_i^k = -\boldsymbol{g}(\boldsymbol{x}^{k-1}, \boldsymbol{V}^{k-1}) \tag{4.5}$$

$$\boldsymbol{A}_i \Delta \boldsymbol{x}_i^k + \boldsymbol{B}_i \Delta \boldsymbol{V}^k = -\boldsymbol{f}_i(\boldsymbol{x}_i^{k-1}, \boldsymbol{V}^{k-1}),\ \ i = 1, \ldots, N. \tag{4.6}$$

which can be detailed as (the iteration superscripts have been ignored for better legibility):

$$\underbrace{\begin{pmatrix} \boldsymbol{D}_1 & \boldsymbol{D}_2 \\ \boldsymbol{D}_3 & \boldsymbol{D}_4 \end{pmatrix}}_{\boldsymbol{D}} \underbrace{\begin{pmatrix} \triangle \boldsymbol{V}^{int} \\ \triangle \boldsymbol{V}^{ext} \end{pmatrix}}_{\triangle \boldsymbol{V}} - \underbrace{\begin{pmatrix} \boldsymbol{0} \\ \sum_{i=1}^{N} \tilde{\boldsymbol{C}}_i \triangle \boldsymbol{x}_i^{ext} \end{pmatrix}}_{\sum_{i=1}^{N} \boldsymbol{C}_i \Delta \boldsymbol{x}_i} = - \underbrace{\begin{pmatrix} \boldsymbol{g}^{int}(\boldsymbol{V}^{int}, \boldsymbol{V}^{ext}) \\ \boldsymbol{g}^{ext}(\boldsymbol{V}^{int}, \boldsymbol{V}^{ext}, \boldsymbol{x}^{ext}) \end{pmatrix}}_{\boldsymbol{g}} \tag{4.7}$$

$$\underbrace{\begin{pmatrix} \boldsymbol{A}_{1i} & \boldsymbol{A}_{2i} \\ \boldsymbol{A}_{3i} & \boldsymbol{A}_{4i} \end{pmatrix}}_{\boldsymbol{A}_i} \underbrace{\begin{pmatrix} \triangle \boldsymbol{x}_i^{int} \\ \triangle \boldsymbol{x}_i^{ext} \end{pmatrix}}_{\triangle \boldsymbol{x}_i} + \underbrace{\begin{pmatrix} \boldsymbol{0} \\ \tilde{\boldsymbol{B}}_i \triangle \boldsymbol{V}^{ext} \end{pmatrix}}_{\boldsymbol{B}_i \Delta \boldsymbol{V}} = - \underbrace{\begin{pmatrix} \boldsymbol{f}_i^{int}(\boldsymbol{x}_i^{int}, \boldsymbol{x}_i^{ext}) \\ \boldsymbol{f}_i^{ext}(\boldsymbol{x}_i^{int}, \boldsymbol{x}_i^{ext}, \boldsymbol{V}^{ext}) \end{pmatrix}}_{\boldsymbol{f}_i} \tag{4.8}$$

where, $\boldsymbol{A}_{1i}$ (resp. $\boldsymbol{D}_1$) accounts for the coupling between the sub-domain's interior variables; $\boldsymbol{A}_{4i}$ (resp. $\boldsymbol{D}_4$) express the coupling between local interface variables; $\boldsymbol{A}_{2i}$ and $\boldsymbol{A}_{3i}$ (resp. $\boldsymbol{D}_2$ and $\boldsymbol{D}_3$) represent the coupling between the local interface and the interior variables; and, $\tilde{\boldsymbol{B}}_i$ (resp. $\tilde{\boldsymbol{C}}_j$) describe the coupling between the local interface variables and the external interface variables of the adjacent sub-domains.

The iterative solution of the above Newton equations gives the values $\boldsymbol{V}(t_n)$ and $\boldsymbol{x}(t_n)$. However, it can be seen that these solution steps are coupled through the interface variables $(\boldsymbol{V}^{ext}, \boldsymbol{x}^{ext})$ and cannot be solved independently. Thus, a Schur-complement approach is used at each iteration to build and solve a reduced system to obtain the interface variables. Once these are computed, the treatment of the N injector sub-domains is decoupled and can be performed independently.

4.4 Schur-complement treatment of interface variables

To treat the interface variables using a Schur-complement approach, the interior variables of each sub-domain need to be eliminated to formulate a reduced system involving only the interface variables (see Section 3.2.3.1).

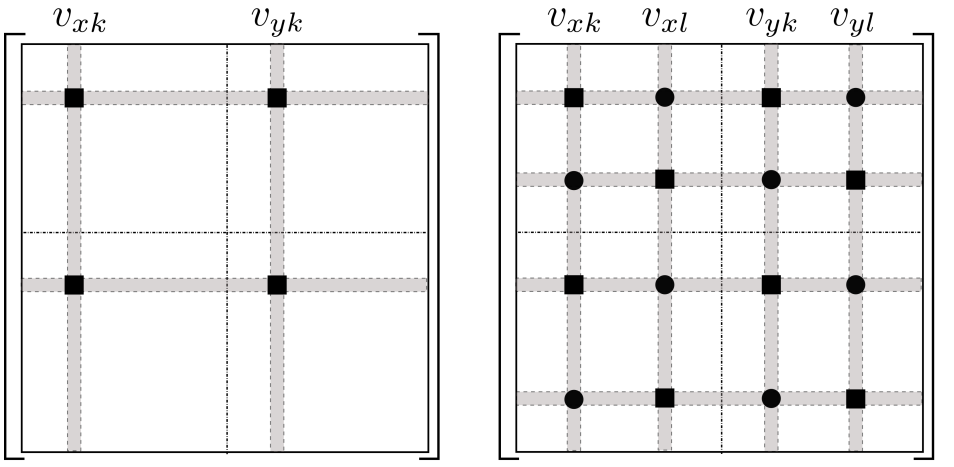
First, the interior variables of the injector sub-domains are eliminated, which yields for the i-th injector ($i = 1, ..., N$):

$$\boldsymbol{S}_i \triangle \boldsymbol{x}_i^{ext} + \tilde{\boldsymbol{B}}_i \triangle \boldsymbol{V}^{ext} = -\tilde{\boldsymbol{f}}_i \tag{4.9}$$

with $\boldsymbol{S}_i = \boldsymbol{A}_{4i} - \boldsymbol{A}_{3i}\boldsymbol{A}_{1i}^{-1}\boldsymbol{A}_{2i}$, the *local* Schur-complement matrix and $\tilde{\boldsymbol{f}}_i = \boldsymbol{f}_i^{ext} - \boldsymbol{A}_{3i}\boldsymbol{A}_{1i}^{-1}\boldsymbol{f}_i^{int}$ the corresponding adjusted mismatch values.

The matrix $\boldsymbol{D}$ of the electric network is very sparse and structurally symmetric [MBB08]. Eliminating the interior variables of the network sub-domain requires building the local Schur-complement matrix $\boldsymbol{S}_D = \boldsymbol{D}_4 - \boldsymbol{D}_3\boldsymbol{D}_1^{-1}\boldsymbol{D}_2$ which in general is *not* a sparse matrix. In addition, $\boldsymbol{S}_D$ is large due to the high number of buses with injectors connected to them. Hence, the computational burden of solving this big and dense matrix can prove prohibitive.

Alternatively, the interior variables of the network sub-domain can be included in the solution of the reduced system, that is $\boldsymbol{S}_D = \boldsymbol{D}$. This approach increases the size of the



(a) Injector attached to k-th bus

(b) Twoport injector attached to k-th and l-th buses

Figure 4.2: Fill-in terms of $\boldsymbol{D}_4$ due to the Schur-complement terms

reduced system by the size of $\boldsymbol{V}^{int}$ but retains sparsity. In this work, this second option was chosen as it allows to use fast sparse linear solvers for the solution of the reduced system.

So, the reduced system to be solved, with the previous considerations, takes on the form:

$$\begin{pmatrix} \boldsymbol{S}_1 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \tilde{\boldsymbol{B}}_1 \\ \boldsymbol{0} & \boldsymbol{S}_2 & \boldsymbol{0} & \cdots & \boldsymbol{0} & \tilde{\boldsymbol{B}}_2 \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S}_3 & \cdots & \boldsymbol{0} & \tilde{\boldsymbol{B}}_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{D}_1 & \boldsymbol{D}_2 \\ -\tilde{\boldsymbol{C}}_1 & -\tilde{\boldsymbol{C}}_2 & -\tilde{\boldsymbol{C}}_3 & \cdots & \boldsymbol{D}_3 & \boldsymbol{D}_4 \end{pmatrix} \begin{pmatrix} \triangle \boldsymbol{x}_1^{ext} \\ \triangle \boldsymbol{x}_2^{ext} \\ \triangle \boldsymbol{x}_3^{ext} \\ \vdots \\ \triangle \boldsymbol{V}^{int} \\ \triangle \boldsymbol{V}^{ext} \end{pmatrix} = - \begin{pmatrix} \tilde{\boldsymbol{f}}_1 \\ \tilde{\boldsymbol{f}}_2 \\ \tilde{\boldsymbol{f}}_3 \\ \vdots \\ \boldsymbol{g}^{int} \\ \boldsymbol{g}^{ext} \end{pmatrix} \tag{4.10}$$

Due to the star layout of the decomposed system, the resulting global Schur-complement matrix is in the so-called *block bordered diagonal form*. Manipulating this structure, which is a unique characteristic of star-shaped layout decomposition, the interface variables of the injector sub-domains can be eliminated and only the variables associated to the network sub-domain remain. This results in the simplified, sparse, reduced system:

$$\underbrace{\begin{pmatrix} \boldsymbol{D}_1 & \boldsymbol{D}_2 \\ \boldsymbol{D}_3 & \boldsymbol{D}_4 + \sum_{i=1}^{N} \tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \tilde{\boldsymbol{B}}_i \end{pmatrix}}_{\widetilde{\boldsymbol{D}}} \underbrace{\begin{pmatrix} \Delta \boldsymbol{V}^{int} \\ \Delta \boldsymbol{V}^{ext} \end{pmatrix}}_{\Delta \boldsymbol{V}} = - \underbrace{\begin{pmatrix} \boldsymbol{g}^{int} \\ \boldsymbol{g}^{ext} + \sum_{i=1}^{N} \tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \tilde{\boldsymbol{f}}_i \end{pmatrix}}_{\widetilde{\boldsymbol{g}}} \tag{4.11}$$

The reduced system (4.11) can be solved efficiently using a sparse linear solver to acquire $\Delta \boldsymbol{V}$. Then, the network sub-domain interface variables are backward substituted in (4.8) and the latter are used to compute the variables $\Delta \boldsymbol{x}_i$ of each injector sub-domain independently.

The nonzero structure of the elimination terms $\tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \tilde{\boldsymbol{B}}_i$ depends on the number of buses the injector is attached to. If it is an injector attached to a single bus, the elimination contributes four elements; the interfacing variables are the two injector current components (i_x, i_y) and the two bus voltage components (v_x, v_y). This term modifies only four, already non-zero, elements of sub-matrix $\boldsymbol{D}_4$, thus retaining its original sparsity pattern. This is shown in Fig. 4.2a for an injector attached to the k-th bus.

On the contrary, if the injector is attached to two buses (twoport), the elimination term $\tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \tilde{\boldsymbol{B}}_i$ contributes with 16 non-zero elements: eight of them modifying already non-zero elements of sub-matrix $\boldsymbol{D}_4$ and the other eight creating a fill-in connecting the two buses. Of course, if the two buses were already connected, for instance with a line or transformer, the aforementioned elements are already non-zero and no fill-in terms are created by the Schur-complement terms. This is shown in Fig. 4.2b for a twoport attached to the k-th and l-th buses.

Even though the twoports might introduce a fill-in to $\widetilde{\boldsymbol{D}}$, the computational burden of solving (4.11) is not significantly affected by them. First, the number of twoports in common power system models is limited compared to their overall size, thus the percentage of fill-ins is also small. Moreover, only sub-matrix $\boldsymbol{D}_4$ is affected by the fill-ins leaving the remaining sub-matrices unaffected. Finally, the sparsity pattern of matrix $\widetilde{\boldsymbol{D}}$ remains the same throughout the entire simulation, thus the sparse matrix structure analysis (optimal ordering) is performed only once.

Similarly, the mismatch correction term $\tilde{\boldsymbol{C}}_i \, \boldsymbol{S}_i^{-1} \, \tilde{\boldsymbol{f}}_i$ contributes only with two elements to $\boldsymbol{g}^{ext}$ for an injector attached to one bus or with four elements for a twoport.

For the sake of generality and to create a link to publications [AFV13a, AV13, AV14b], Eq. 4.11 can be rewritten as:

$$(\boldsymbol{D} + \sum_{i=1}^{N} \boldsymbol{C}_i \boldsymbol{A}_i^{-1} \boldsymbol{B}_i) \, \Delta \boldsymbol{V}^k = - \, \boldsymbol{g}(\boldsymbol{x}^{k-1}, \boldsymbol{V}^{k-1}) - \sum_{i=1}^{N} \boldsymbol{C}_i \boldsymbol{A}_i^{-1} \boldsymbol{f}_i(\boldsymbol{x}_i^{k-1}, \boldsymbol{V}^{k-1}) \tag{4.12}$$

where:

$$\begin{aligned} \boldsymbol{C}_i \boldsymbol{A}_i^{-1} \boldsymbol{B}_i &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{C}}_i \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_{1i} & \boldsymbol{A}_{2i} \\ \boldsymbol{A}_{3i} & \boldsymbol{A}_{4i} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{B}}_i \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{C}}_i \left(\boldsymbol{A}_{4i} - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \right)^{-1} \tilde{\boldsymbol{B}}_i \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \tilde{\boldsymbol{B}}_i \end{bmatrix} \end{aligned}$$

and:

$$
\begin{aligned}
\boldsymbol{C}_i \boldsymbol{A}_i^{-1} \boldsymbol{f}_i &= \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tilde{\boldsymbol{C}}_i \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_{1i} & \boldsymbol{A}_{2i} \\ \boldsymbol{A}_{3i} & \boldsymbol{A}_{4i} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{f}^{int}{}_i \\ \boldsymbol{f}^{ext}{}_i \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ -\tilde{\boldsymbol{C}}_i \left(\boldsymbol{A}_{4i} - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \right)^{-1} \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} & \tilde{\boldsymbol{C}}_i \left(\boldsymbol{A}_{4i} - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \right)^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{f}^{int}{}_i \\ \boldsymbol{f}^{ext}{}_i \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0} \\ \tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \left(\boldsymbol{f}^{ext}{}_i - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{f}^{int}{}_i \right) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0} \\ \tilde{\boldsymbol{C}}_i \boldsymbol{S}_i^{-1} \tilde{\boldsymbol{f}}_i \end{bmatrix}
\end{aligned}
$$

In both derivations, the following formula [BIG76] was used:

$$
\begin{bmatrix} \boldsymbol{A}_{1i} & \boldsymbol{A}_{2i} \\ \boldsymbol{A}_{3i} & \boldsymbol{A}_{4i} \end{bmatrix}^{-1} = \begin{bmatrix} \left(\boldsymbol{A}_{1i} - \boldsymbol{A}_{2i} \boldsymbol{A}_{4i}^{-1} \boldsymbol{A}_{3i} \right)^{-1} & -\boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \left(\boldsymbol{A}_{4i} - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \right)^{-1} \\ -\left(\boldsymbol{A}_{4i} - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \right)^{-1} \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} & \left(\boldsymbol{A}_{4i} - \boldsymbol{A}_{3i} \boldsymbol{A}_{1i}^{-1} \boldsymbol{A}_{2i} \right)^{-1} \end{bmatrix}
$$

While both Eqs 4.11 and 4.12 are equivalent, the latter doesn't make any assumption on the ordering of the interior and interface variables. Hence, the elimination of the interior variables shown in (4.9) is implied through the proper selection of matrices $\boldsymbol{C}_i$ and $\boldsymbol{B}_i$.

4.5 Parallel algorithm

As discussed in Chapter 3, the main reason to employ a DDM is the parallelization opportunities inherent to this type of algorithms. The overall solution algorithm is sketched in Fig. 4.3 with the parallel segments being shaded. For each discrete time instant t_n, the parallel DDM described below is used to solve the sub-systems and obtain $\boldsymbol{V}(t_n)$ and $\boldsymbol{x}(t_n)$.

First, the sub-domain local systems (4.7) and (4.8) are updated, their matrices are factorized, and their contributions to the reduced system (4.11) are computed. The sub-systems are processed in parallel as there are no data dependencies between them during the update procedure. This can be seen in *BLOCK A* of Fig. 4.3.

Then, the reduced system (4.11), whose consisting elements were computed in parallel, is solved in *BLOCK B* to obtain the voltage corrections $\Delta \boldsymbol{V}$. The solution is performed by a sparse linear solver and its computational burden amounts for 5-8% of the total simulation time (more information will be given in Section 4.8.2). This is an unavoidable sequential bottleneck introduced by Schur-complement-based methods and can possibly impose a limit to the algorithm's scalability (see Section 2.5.2).

Next, the voltage corrections are introduced into the injector sub-systems (4.8), thus decoupling them. The latter are then solved in parallel, as shown in *BLOCK C* of Fig. 4.3, to obtain the injector variable corrections $\Delta \boldsymbol{x}_i$. At the end of this block, both the voltage vector ($\boldsymbol{V}$) and DAE states ($\boldsymbol{x}$) have been updated.

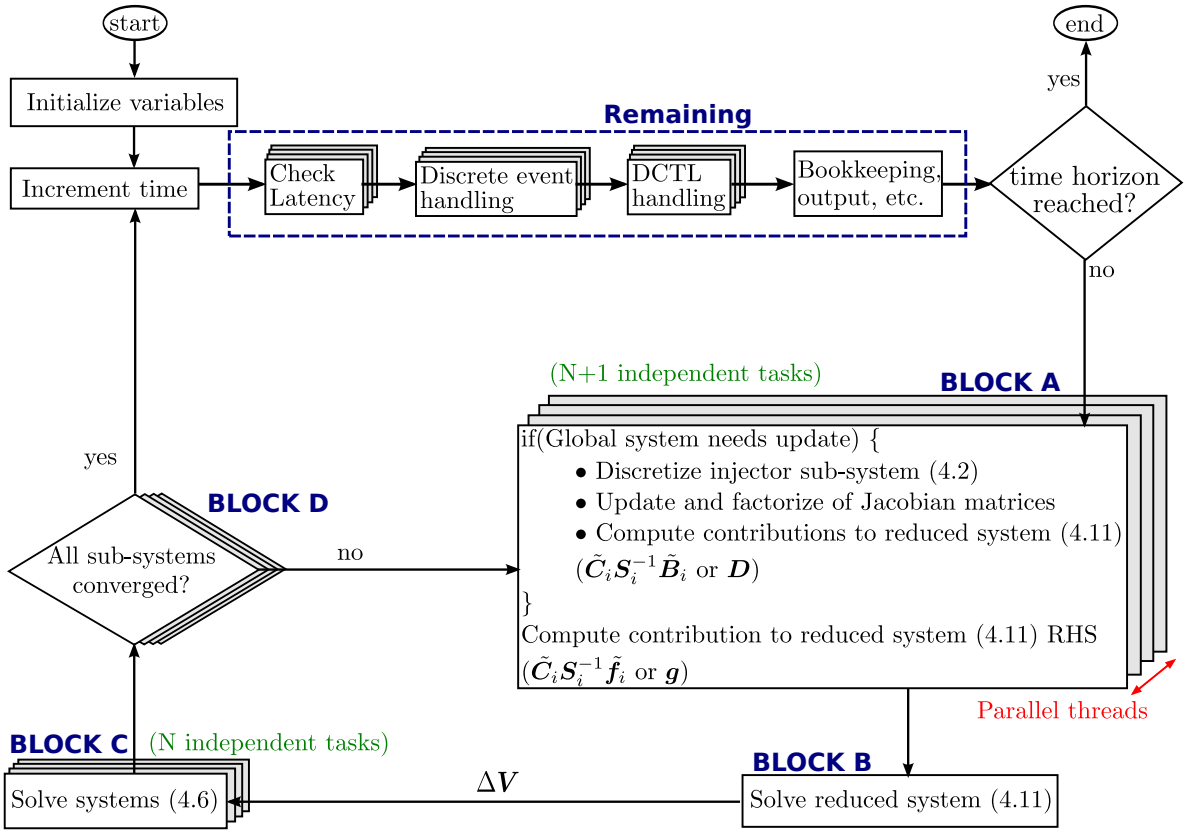


Figure 4.3: Parallel solution algorithm

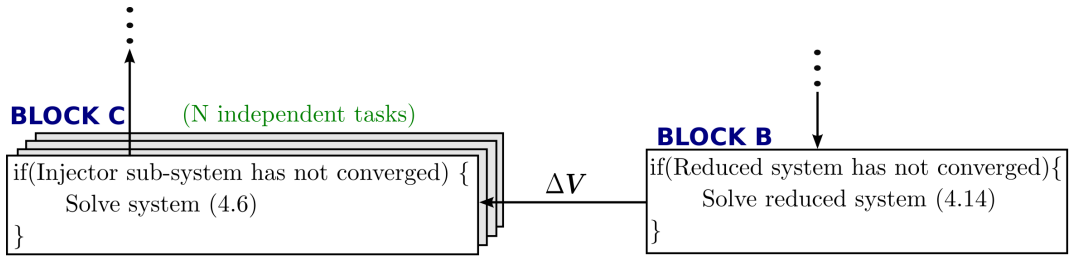


Figure 4.4: *BLOCK B and C* with skipping converged sub-systems

Finally, the convergence of the sub-systems is checked in parallel in *BLOCK D* using the updated voltage and state vectors. If all sub-systems have converged, the algorithm proceeds to the next time instant, otherwise, a new parallel solution is initiated.

4.6 Localization techniques

The concept of localization results from the observation that in large power systems a disturbance often affects a small number of components while the remaining are only slightly influenced [Bra93]. Similar observations have been made in other fields [HSV81]. In this work, this fact is exploited in three ways. These techniques are made possible due to the decomposition of the system that allows to detect and treat individually the sub-domains.

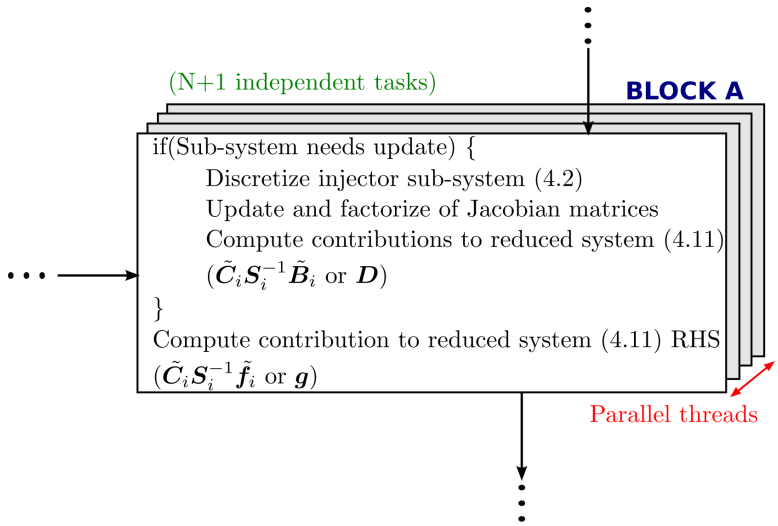


Figure 4.5: *BLOCK A* with asynchronous update of sub-domain matrices

4.6.1 Skipping converged sub-systems

First, this technique is used within one discretized time instant solution to stop computations of injectors (resp. reduced system) who have already been solved with the desired tolerance. That is, after one decomposed solution of (4.5) and (4.6), the convergence of each injector and the reduced system is checked individually. If the convergence criterion is satisfied, then the specific sub-systems are flagged as converged. For the remaining iterations of the current time instant, the sub-system is not solved, although its mismatch, computed with (4.4) or (4.11), is monitored to guarantee that it remains converged. This technique decreases the computational effort within one discretized time instant without affecting the accuracy of the solution (see Sections 4.7 and 4.9). Thus, *BLOCKS B and C* in Fig. 4.3 are replaced by the block in Fig. 4.4.

From a mathematical point of view, the state corrections of converged injectors are set to zero ($\Delta \boldsymbol{x}_i = 0$), thus the RHS $\boldsymbol{f}_i$ of (4.6) and the sensitivity to voltage deviations $\boldsymbol{B}_i$ are set to zero. At the same time, for a converged reduced system the voltage corrections are set to zero ($\Delta \boldsymbol{V} = \boldsymbol{0}$), thus in Eq. 4.11 the RHS $\widetilde{\boldsymbol{g}}$ is set to zero.

4.6.2 Asynchronous update of sub-domain matrices

Taking advantage of the fact that each sub-domain is solved by a separate quasi-Newton method (in this work, the VDHN presented in Section 1.2.5.2), the sub-system update criteria are decoupled and their local matrices (such as $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{D}$), as well as their Schur-complement terms, are updated asynchronously. In this way, sub-domains which converge fast keep the same local system matrices for many iterations and even time-steps, while sub-domains which converge slower update their matrices more frequently. Thus, *BLOCK A* in Fig. 4.3 is replaced by the block in Fig. 4.5.

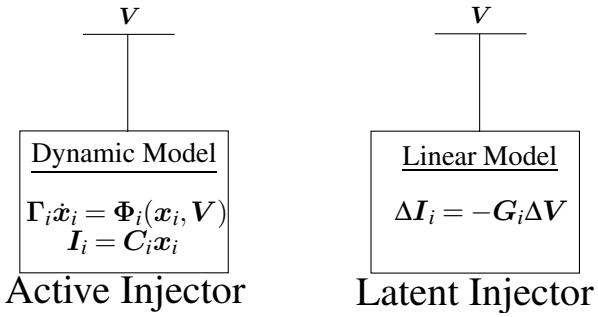


Figure 4.6: Active VS latent injector models

Traditional update criteria of VDHN methods are reused to trigger the sub-system matrices computation. That is, if the sub-system has not converged after five iterations of the algorithm presented in Section 4.5, the local matrices are updated. Moreover, an update of the matrices is triggered if a change of the sub-system equations, caused by a discrete event, is detected. Of course, after a severe event in the system (such a short-circuit, the tripping of a generator, etc.) or when the time step used for the discretization of Eqs. 4.2 is changed, an update of all the matrices and the Schur-complement terms is forced to avoid convergence problems.

4.6.3 Latency

Localization is also exploited over several time steps by detecting, during the simulation, the injectors marginally participating to the system dynamics (*latent*) and replacing their dynamic models (4.2) with much simpler and faster to compute sensitivity-based models. At the same time, the full dynamic model is used if an injector exhibits significant dynamic activity (*active*). The two models are shown in Fig. 4.6.

The sensitivity-based model is derived from the linearized Eqs. 4.6 when ignoring the internal dynamics, that is $\boldsymbol{f}_i(\boldsymbol{x}_i^{k-1}, \boldsymbol{V}^{k-1}) \simeq \boldsymbol{0}$, and solving for the state variation $\Delta \boldsymbol{x}_i$:

$$\Delta \boldsymbol{x}_i \simeq -\boldsymbol{A}_i^{-1} \boldsymbol{B}_i \Delta \boldsymbol{V}$$

The corresponding current variation $\Delta \boldsymbol{I}_i$ is given by:

$$\Delta \boldsymbol{I}_i = -\boldsymbol{E}_i \boldsymbol{A}_i^{-1} \boldsymbol{B}_i \Delta \boldsymbol{V} = -\boldsymbol{G}_i \Delta \boldsymbol{V} \tag{4.13}$$

where $\boldsymbol{E}_i$ (similarly to $\boldsymbol{C}_i$) is a trivial matrix with zeros and ones whose purpose is to extract the injector current variations from $\Delta \boldsymbol{x}_i$ and $\boldsymbol{G}_i$ is the sensitivity matrix relating the current with the voltage variation.

Selecting an arbitrary instant t^*, the linear relation (4.13) can be rewritten as:

$$\boldsymbol{I}_i(t_n) = \boldsymbol{I}_i(t^*) - \boldsymbol{G}_i(t^*) \left[\boldsymbol{V}(t_n) - \boldsymbol{V}(t^*)\right] \tag{4.14}$$

for any discrete time $t_n \geq t^*$.

The linear model (4.14) is a valid estimate of the full dynamic model (4.2) only when the injector shows low dynamic activity (thus, the RHS of Eq. 4.6 can be neglected) and only for small deviations around the linearization point (thus, $\boldsymbol{G}_i$ can be considered constant). However, as it will be shown later on, this technique can introduce some error into the simulated response.

It is important to note that, the Schur-complement term contributed by the linear model (4.14) to matrix $\tilde{\boldsymbol{D}}$ of Eq. 4.11 *is the same as the one of model* (4.6). This means that switching from one model to the other doesn't require to recompute and factorize the Schur-complement matrix $\tilde{\boldsymbol{D}}$.

4.6.3.1 Monitoring variable and metrics

The essence of the algorithm lies in its ability to detect the switching of injectors from active to latent, and conversely. During the dynamic simulation the state vector values $\boldsymbol{x}(t_j)$ and $\boldsymbol{V}(t_j)$ are known for $t_1, ..., t_j, ..., t_n$, with t_n the last computed discrete time. The injector switching criteria have to be robust and based only on currently available information. Furthermore, as the algorithm aims for higher simulation performance, the criteria computations need to be fast and use as little memory as possible.

Since the injectors interact with the network and between them through the current and voltage changes (see Eq. 4.3), power flow variations can be used as an indication of dynamic activity. Therefore, the variation of the per-phase apparent power ($S_i = \sqrt{{P_i}^2 + {Q_i}^2}$) flowing in each injector was naturally selected as the monitoring variable representative of the injector dynamic activity. Alternatively, more detailed information could be extracted from the active (P_i) and reactive (Q_i) powers but at the cost of doubling the computing effort and memory usage.

Simply stated, an injector is declared latent when its apparent power S_i has "not changed significantly for some time" or, in other words, exhibits small variability. The S_i values are available as time-series samples. Thus, traditional methods for analyzing time series data can be employed to characterize the variability of S_i over a pre-specified, moving, time window (T_L). This procedure is shown in Fig. 4.7.

The choice of using a moving time window and not the entire history aims at disregarding the oldest "behavior" of an injector and involving only recently observed dynamics. However, if the time window is very small, smooth variations (i.e. with low rates of change) may not be detected.

The main characteristics extracted from the time-series are the sample average value ($S_{i,av}$), variance ($S_{i,var}$), and standard deviation ($S_{i,std}$). In particular, the standard deviation is the measure of volatility that shows how much variation or dispersion exists from the average. A small standard deviation indicates that the data points tend to be very close to the average, whereas high standard deviation indicates that the data points are spread out over a large range of values. Consequently, standard deviation of S_i smaller than a selected tolerance

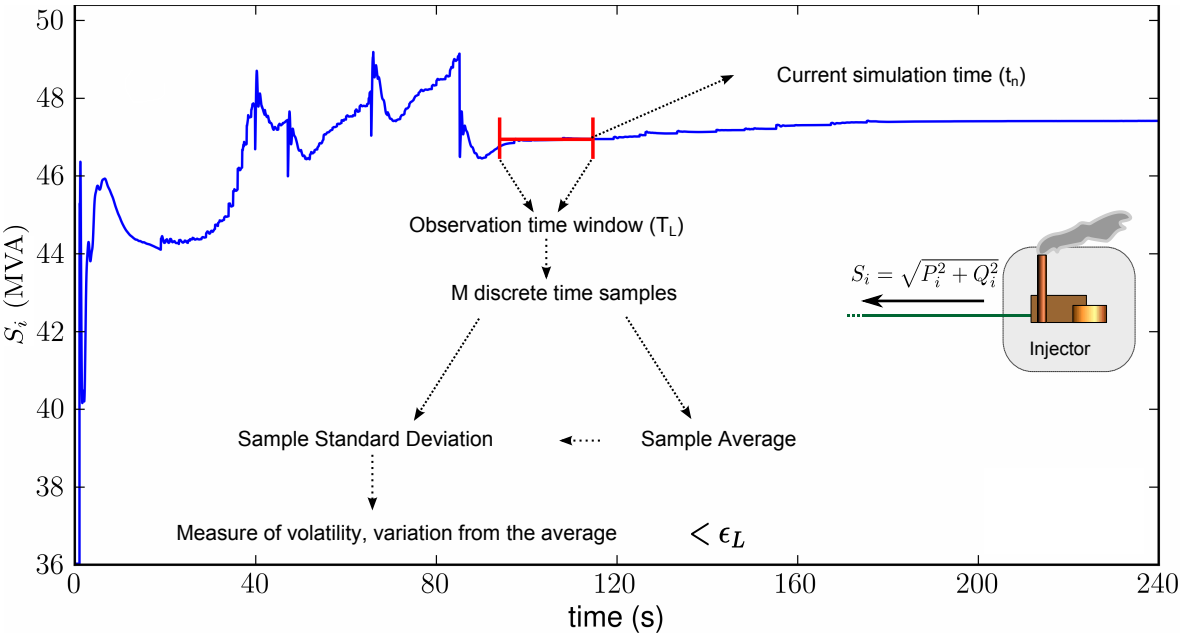


Figure 4.7: Monitoring injector apparent power to declare latent

ϵ_L is an indication that the i-th injector exhibits low dynamic activity and can be considered latent.

After an injector is classified as latent at time t^*, the apparent power is no longer dictated by the dynamic model (4.2) but by the linear model (4.14), affected only by the deviation of the voltage. Therefore, the standard deviation is not reliable to switch the injector back to active mode since slow voltage changes can gradually "drift" the injector's operating point away from the reference without the standard deviation ever increasing. To avoid this, the *absolute deviation* of the apparent power from its reference value $S_i(t^*)$ is used. If the absolute variation is bigger than the same tolerance ϵ_L, meaning that the model has moved away from the linearization point, the injector returns to active mode.

4.6.3.2 Exponential moving average

Large power systems may involve thousands of injectors, making it inefficient to keep all injector apparent powers in memory over the moving time window. For example, a small time window $T_L = 5$ s has 250 points when the system simulated with a time step of 20 ms. Thus, in the Hydro-Québec system of Section 1.3, which has 4601 injectors, this translates to more than one million points that need to be kept in memory and updated at every time step.

Furthermore, calculating the exact average and standard deviation values of the time-window samples at each time-step leads to complex bookkeeping and time consuming computations. To avoid this, an approximation is considered, which originates from real-time digital signal processing where computing and memory resources are scarce. When a new sample $S_i(t_n)$ is available, a moving average value and standard deviation can be computed with a

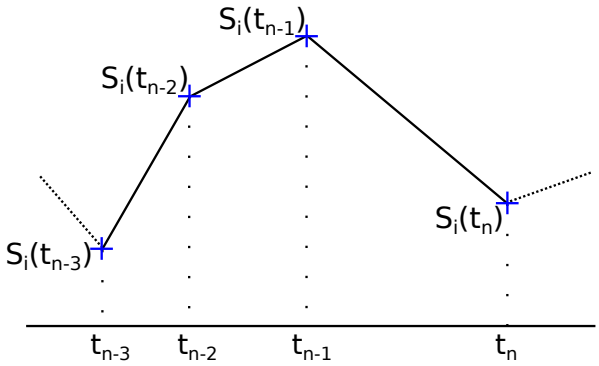


Figure 4.8: Linear variation of S_i values

repeated application of the Exponential Moving Average (EMA) [Mul01]. This is a weighted moving average with the weights decreasing exponentially. Each sample is valued some percent smaller than the next more recent sample. With this constraint the moving average can be computed very efficiently.

There are several definitions of EMA [Eck13]. Due to the fact that the time step $h_n = t_n - t_{n-1}$ between the samples could vary, the weight of each sample changes according to the fraction of the time step over the observation time window T_L. Thus, the EMA operator is defined recursively as [Mul01]:

$$S_{i,av}(t_n) = EMA\left(\boldsymbol{S}_i, T_L, h_n\right) = \lambda_1 S_{i,av}(t_{n-1}) + (1-\lambda_1) S_i(t_n) + (\lambda_1 - \lambda_2)\left(S_i(t_n) - S_i(t_{n-1})\right) \tag{4.15}$$

where $\boldsymbol{S}_i$ includes the last two values of S_i, $\alpha = \dfrac{h_n}{T}$ and $\lambda_1 = e^{-\alpha}$. The variable λ_2 depends on the series type and the interpolation method selected:

$$\lambda_2 = \begin{cases} \dfrac{1-\lambda_1}{\alpha} & \text{for the linear interpolation} \\ 1 & \text{for taking the preceding series value} \\ \sqrt{\lambda_1} & \text{for taking the nearest series value} \\ \lambda_1 & \text{for taking the subsequent series value} \\ \lambda_1 & \text{for equally spaced discrete time series (no interpolation)} \end{cases} \tag{4.16}$$

In this work, it is assumed that the value of S_i varies linearly between any two successive time-steps as shown in Fig. 4.8, thus $\lambda_2 = \dfrac{1-\lambda_1}{\alpha}$.

Even though the exponential moving variance can be obtained as [HWYC09]:

$$S_{i,var}(t_n) = EMA(\boldsymbol{S}_i^2, T_L, h_n) - \left(EMA(\boldsymbol{S}_i, T_L, h_n)\right)^2 = EMA(\boldsymbol{S}_i^2, T, h_n) - S_{i,av}^2(t_n) \tag{4.17}$$

the same reasoning as for formula (4.15) can be also used [Mul01]:

$$S_{i,var}(t_n) = \lambda_1 S_{i,var}(t_{n-1}) + (1-\lambda_1)\Delta S_i^2(t_n) + (\lambda_1 - \lambda_2)\left(\Delta S_i^2(t_n) - \Delta S_i^2(t_{n-1})\right) \tag{4.18}$$

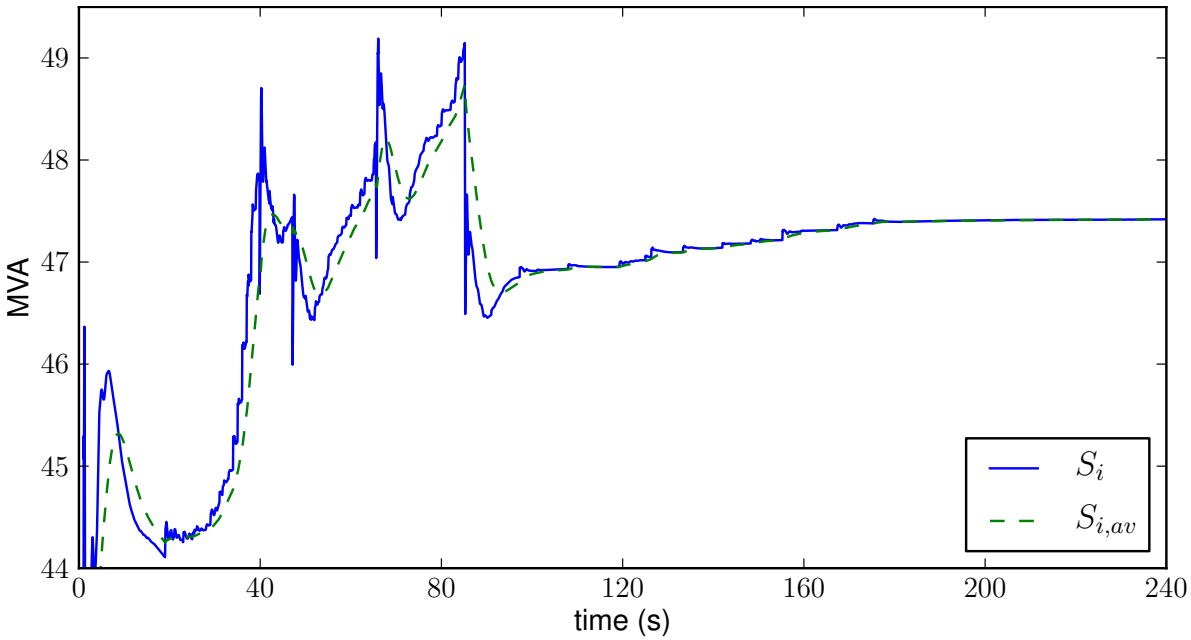


Figure 4.9: S_i and $S_{i,av}$ of a synchronous machine after a fault

where $\Delta S_i(t_n) = S_i(t_n) - S_{i,av}(t_n)$. Both formulas were tested in RAMSES and no significant differences were found. Equation 4.18 is used for the investigation following.

Finally, the standard deviation can be obtained by:

$$S_{i,std}(t_n) = \sqrt{S_{i,var}(t_n)} \tag{4.19}$$

This value can be used to assess the volatility and thus the dynamic activity of an injector. For example, Fig. 4.9 shows the long-term evolution of the apparent power and corresponding exponential moving average of a synchronous machine, after the occurrence of a three-phase short-circuit near the generator, which was cleared after five cycles (100 ms).

The observation time window of $T_L = 5$ s used for Fig. 4.9 was found to be the smallest T_L providing smooth averaging. Using smaller time windows with the same time-steps increases the weight of the new value compared to the older ones. During short-term dynamics, it is imperative to use the full injector model that can capture the fast transients. However, a small T_L can lead injectors to switch latent during this period and introduce mistakes in the simulated response. In the cases studied later on, a time-window of $T_L = 10$ s is used as it provides a compromise between speed of response of $S_{i,std}$ and consideration for recent dynamic activity.

4.6.3.3 Switching algorithm

The decision for switching between active and latent mode is taken after solving the system equations for each time step t_n. Then, the selected models are used for the computation of the states at t_{n+1}. During the DDM iterations, the state of each injector (latent or active) does

not change, as switching could perturb the Newton iterations and cause divergence. The complete procedure is given by the following algorithm:

Algorithm 4.1 Injector switching criterion at discrete time t_n

if injector *i* is active **then**
  Calculate $S_i(t_n)$ using nonlinear model (4.2)
  Calculate $S_{i,av}(t_n)$ and $S_{i,std}(t_n)$ using Eqs. 4.15, 4.18 and 4.19
  if $S_{i,std}(t_n) \leq \epsilon_L$ **then**
    $t^* = t_n$
    injector *i* ←latent
  end if
else
  Calculate $S_i(t_n)$ using linear model (4.14)
  if $\mid S_i(t_n) - S_{i,av}(t^*) \mid \geq \epsilon_L$ **then**
    injector *i* ←active
  end if
end if

An example of the switching procedure is shown in Fig. 4.10. In the upper plot, the $S_{i,std}$ is shown along with the latency threshold ϵ_L, here set to 0.1 MVA. At time $t = 107$ s, the threshold is crossed and the injector switched to latent. In the lower plot, it can be seen that the apparent power of the latent injector remains within the deadband centered on the linearization point, until $t = 125$ s when it is switched back to active mode.

If latency is used during the simulation in combination with the previous technique, the *BLOCK C* of Fig. 4.3 is replaced by the block of Fig. 4.11.

The parameter ϵ_L controls the approximation introduced into the simulation ($\epsilon_L = 0$ results in fully accurate simulation). If the power system includes injectors of both very small and very large powers then ϵ_L must remain small to keep the error bounded. On the other hand, if the system involves similarly sized injectors, then ϵ_L can be increased without introducing large errors.

Figure 4.12 shows the apparent power of a synchronous machine with latency $\epsilon_L = 0$ MVA and $\epsilon_L = 0.1$ MVA applied to the entire system. The portions with gray background denote the areas when the pictured generator has turned latent. The corresponding relative error on the simulated evolution is shown in Fig. 4.13. It can be seen that the inaccuracy introduced by latency is negligible, and it is so for most dynamic simulations.

The injector shown in Fig. 4.12 switches to latent for first time at $t \approx 130$ s. However, other injectors in the system get latent earlier. This is the reason for the error shown in Fig. 4.13 prior to $t \approx 130$ s.

From a mathematical point of view, the effect of latency to the convergence of the simulations will be examined in the next section; while, a more practical analysis of the error introduced, the simulation performance and the proper selection of parameter ϵ_L will be given in Section 4.9.

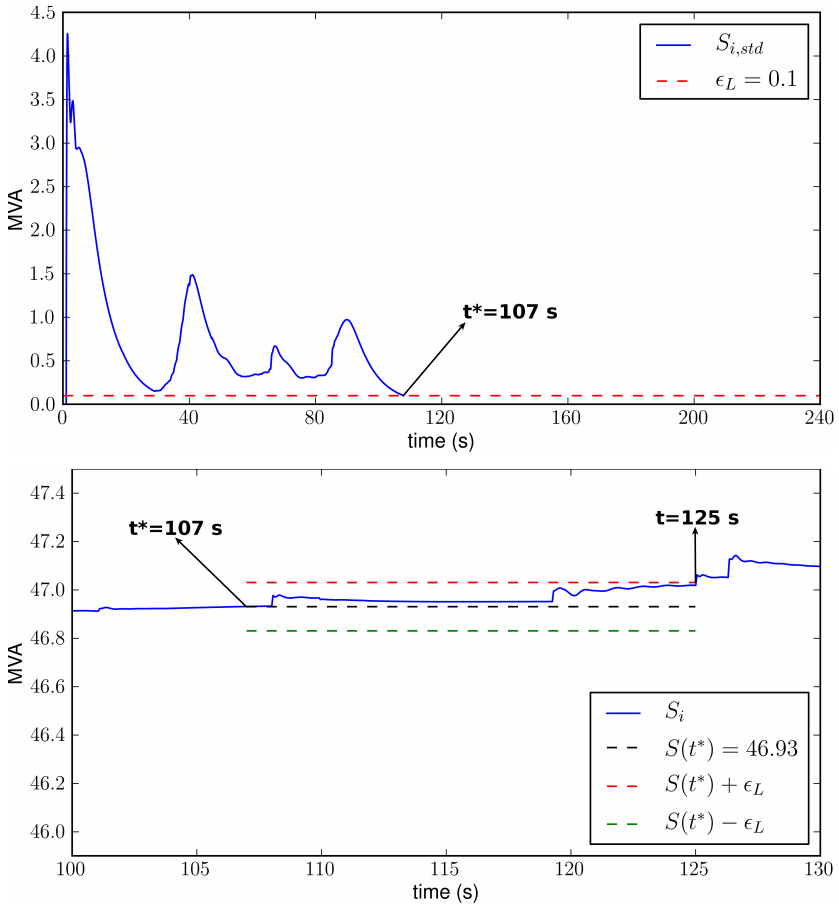


Figure 4.10: Switching procedure between latent and active modes

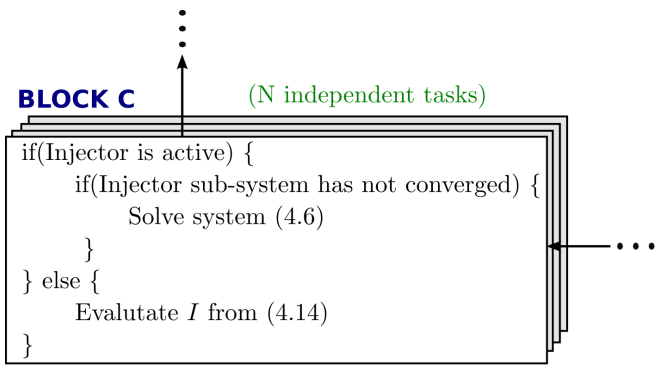


Figure 4.11: *BLOCK C* with skipping converged injectors and latency techniques

4.7 Effects of localization techniques on convergence

By using the Schur-complement approach detailed in Section 4.4, an exact solution of Eqs. 4.5-4.6 is performed at each iteration of the parallel algorithm of Fig. 4.3 (*BLOCKS A-D*). If these equations are grouped a unique linear system, the following equivalent integrated system is

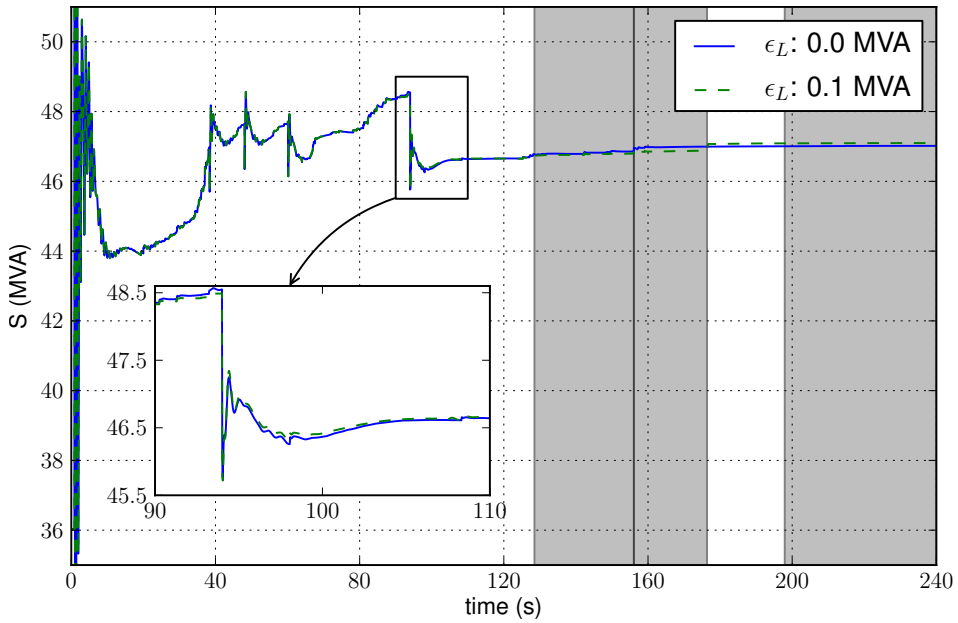


Figure 4.12: Apparent power of synchronous machine with and without latency

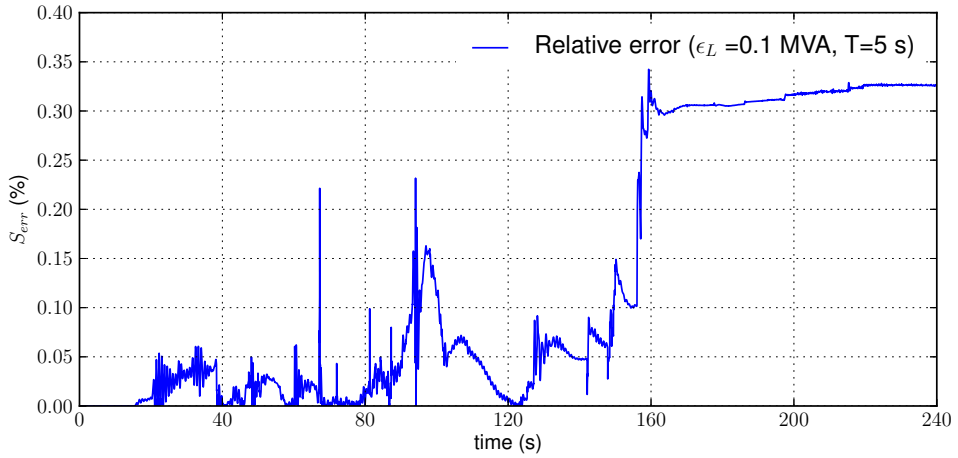


Figure 4.13: Relative error of apparent power output

formulated for the k-th iteration:

$$\underbrace{\begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{B}_1 \\ \boldsymbol{0} & \boldsymbol{A}_2 & \boldsymbol{0} & \cdots & \boldsymbol{B}_2 \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_3 & \cdots & \boldsymbol{B}_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{C}_1 & -\boldsymbol{C}_2 & -\boldsymbol{C}_3 & \cdots & \boldsymbol{D} \end{pmatrix}^k}_{\boldsymbol{J}^k} \underbrace{\begin{pmatrix} \triangle \boldsymbol{x}_1 \\ \triangle \boldsymbol{x}_2 \\ \triangle \boldsymbol{x}_3 \\ \vdots \\ \triangle \boldsymbol{V} \end{pmatrix}^k}_{\triangle \boldsymbol{y}^k} = - \underbrace{\begin{pmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \vdots \\ \boldsymbol{g} \end{pmatrix}^k}_{\boldsymbol{F}^k} \tag{4.20}$$

This system is the same as the linear system (1.15), solved by a classical simultaneous solution presented in Section 1.2.5.2, using the same discretization scheme as the DDM

and performing some row and column permutations. That is, the DDM-based solution presented in this chapter is mathematically equivalent to solving together the set of non-linear, discretized Eqs. 4.3 and 4.4 with a quasi-Newton method.

This equivalence between the DDM-based and the simultaneous approach, allows using the extensive theory behind quasi-Newton schemes to assess the proposed algorithm's convergence. Under some well-studied requirements [Bro70, BDM73, DM74, GT82, Mor99], the iterative Newton schemes converge to the solution at a super-linear rate.

However, the localization techniques modify the equivalent system (4.20). First, the matrices of each injector and of the network are not updated synchronously but according to the local convergence of each sub-system (see Section 4.6.2). Thus, modifying the system into:

$$\begin{pmatrix} \boldsymbol{A}_1^{k_1} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{B}_1^{k_1} \\ \mathbf{0} & \boldsymbol{A}_2^{k_2} & \mathbf{0} & \cdots & \boldsymbol{B}_2^{k_2} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{A}_3^{k_3} & \cdots & \boldsymbol{B}_3^{k_3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{C}_1^{k_{N+1}} & -\boldsymbol{C}_2^{k_{N+1}} & -\boldsymbol{C}_3^{k_{N+1}} & \cdots & \boldsymbol{D}^{k_{N+1}} \end{pmatrix}^k \begin{pmatrix} \triangle \boldsymbol{x}_1 \\ \triangle \boldsymbol{x}_2 \\ \triangle \boldsymbol{x}_3 \\ \vdots \\ \triangle \boldsymbol{V} \end{pmatrix}^k = - \begin{pmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \vdots \\ \boldsymbol{g} \end{pmatrix}^k \tag{4.21}$$

where $k_i \leq k$ $(i = 1, ..., N+1)$ is the iteration at which the i-th injector or the network matrices were last updated. It should be noted that the matrices are often unchanged over several time instants, not only iterations. Thus, this modified system can be treated a quasi-Newton method with a special Jacobian update scheme. The error introduced to the Jacobian by the asynchronous update of the sub-domain matrices is minimal, and while it can affect the convergence rate of the Jacobian, it does not affect the final solution.

Next, by also considering the skip-converged and latency techniques (see Sections 4.6.1 and 4.6.3) the system (4.21) is modified to:

$$\begin{pmatrix} \boldsymbol{A}_1^{k_1} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{B}_1^{k_1} \\ \mathbf{0} & \boldsymbol{A}_2^{k_2} & \mathbf{0} & \cdots & \boldsymbol{B}_2^{k_2} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{A}_3^{k_3} & \cdots & \boldsymbol{B}_3^{k_3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{C}_1^{k_{N+1}} & -\boldsymbol{C}_2^{k_{N+1}} & -\boldsymbol{C}_3^{k_{N+1}} & \cdots & \boldsymbol{D}^{k_{N+1}} \end{pmatrix}^k \begin{pmatrix} \triangle \boldsymbol{x}_1 \\ \triangle \boldsymbol{x}_2 \\ \triangle \boldsymbol{x}_3 \\ \vdots \\ \triangle \boldsymbol{V} \end{pmatrix}^k = - \begin{pmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \vdots \\ \boldsymbol{g} \end{pmatrix}^k + \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \boldsymbol{r}_3 \\ \vdots \\ \boldsymbol{r}_{N+1} \end{pmatrix}^k \tag{4.22}$$

That is, skipping the solution of a converged injector or the reduced system is equivalent to setting to zero the off-diagonal blocks and the mismatches of (4.22). In addition, when an injector is considered latent, it is equivalent to setting $\boldsymbol{r}_i = \boldsymbol{f}_i$. Thus, both techniques consist in setting:

$$\boldsymbol{r}_i = \begin{cases} \boldsymbol{f}_i & \text{if latent or converged} \\ \mathbf{0} & \text{otherwise} \end{cases} \qquad \boldsymbol{B}_i^{k_i} = \begin{cases} \mathbf{0} & \text{if converged} \\ \boldsymbol{B}_i^{k_i} & \text{otherwise} \end{cases}$$

$$\boldsymbol{r}_{N+1} = \begin{cases} \boldsymbol{g} & \text{if converged} \\ \boldsymbol{0} & \text{otherwise} \end{cases} \qquad \boldsymbol{C}_i^{k_{N+1}} = \begin{cases} \boldsymbol{0} & \text{if converged} \\ \boldsymbol{C}_i^{k_{N+1}} & \text{otherwise} \end{cases}$$

These changes lead to an inexact Newton scheme and its convergence properties and assumptions can be examined as shown in Appendix A.

Comparing the skip-converged and latency techniques, it can be seen that the former applies more strict criteria and does not affect the final solution of the algorithm. More specifically, in RAMSES if the i-th injector has converged and is not solved anymore, its mismatch is still computed at each iteration and r_i^k is updated. If at any iteration r_i^k increases compared to r_i^{k-1}, then the injector is solved again. Thus, relating to Eq. A.11, if skip-converged is used alone, the following condition can be easily checked to ensure the convergence:

$$\frac{\|\boldsymbol{r}^k\|}{\|\boldsymbol{F}(\boldsymbol{y}^k)\|} < \eta < 1 \qquad \forall k \geq 0 \tag{4.23}$$

On the contrary, the latency technique does not rely on numerical criteria but rather on observations of the component dynamic response. The methods detailed in Section 4.6.3.2 to switch between active and latent injectors are approximate. Although a small standard deviation of the apparent power output of an injector is an indication of low dynamic activity, it does not guarantee that the equivalent model used will provide *exactly* the same results as the full dynamic model. Especially when a large latency tolerance is used, the error introduced in the system (4.22) could affect the accuracy of the solution.

4.8 Parallelization specifics

As discussed in Chapter 2, for the parallel implementation of the algorithm the OpenMP API has been used. The shaded blocks shown in Fig. 4.3 have been parallelized using the fork-join pattern. That is, when the *master* computational thread reaches the shaded block, it forks into M computational threads (where M is selected by the user or set equal to the number of cores). When all computations within the block are finished, the threads join again. This procedure for one iteration of the DDM-based algorithm is shown in Fig. 4.14, in a fictitious example with $N = 8$ injectors, and $M = 1$ or $M = 4$ threads.

The squares in the figure denote the independent tasks within each algorithmic block. That is, in *BLOCK A* the red square denotes an update of the sub-domain local matrices and the Schur-complement terms; in *BLOCK B*, the solution of the Schur-complement system; in *BLOCK* C, the solution of the injector sub-systems; and, in *BLOCK* D, the convergence check. Let us assume, for reasons of simplicity, that i) the computational burden of each red square is the same and equal to one; ii) the work is shared in the best possible way among the threads (minimum imbalance); and, iii) there is no OverHead Cost (OHC) related to the fork and join procedures. Then, according to Amdahl's law (Section 2.5.2) scalability can be

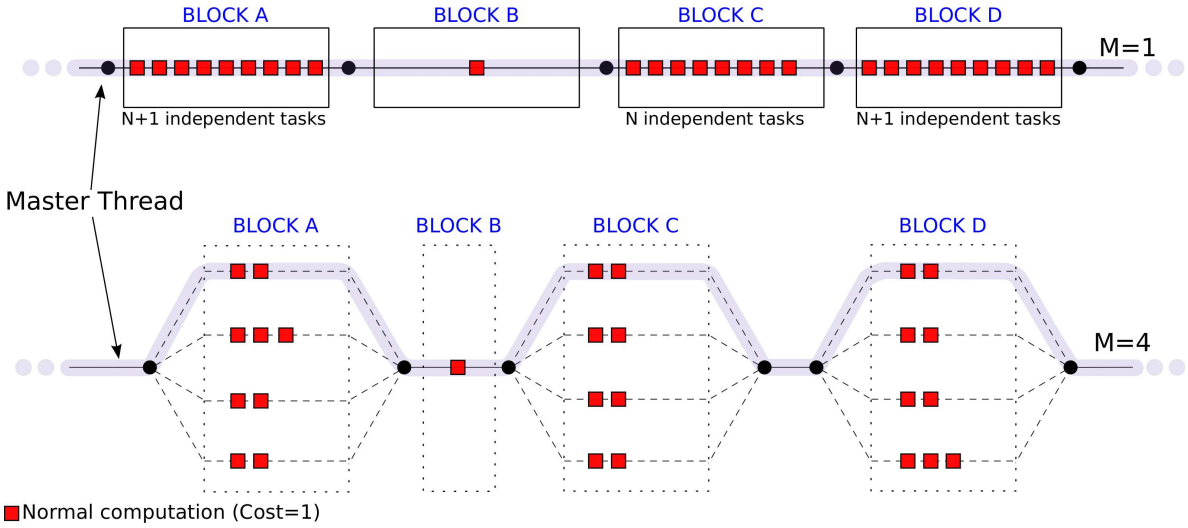


Figure 4.14: Fork-join pattern for parallelization of algorithm in Fig. 4.3

computed as:

$$Scalability_4 = \frac{T_S + T_P}{T_S + \frac{T_P}{4}} = \frac{T_B + T_A + T_C + T_D}{T_B + \frac{T_A}{4} + \frac{T_C}{4} + \frac{T_D}{4}} = \frac{1 + 9 + 8 + 9}{1 + \frac{9}{4} + \frac{8}{4} + \frac{9}{4}} = 3.6 \tag{4.24}$$

where $T_S + T_P = T_1 = 27$ is the sequential execution time ($M = 1$) and $T_S + \frac{T_P}{4} = 7.5$ is the expected execution time on four threads. The maximum scalability can be computed with:

$$Scalability_\infty = \lim_{M \to \infty} \left(\frac{T_S + T_P}{T_S + \frac{T_P}{M}} \right) = \frac{27}{1} = 27 \tag{4.25}$$

On the other hand, according to the work-span model (Section 2.5.4):

$$Scalability_4 = \frac{T_1}{T_4} = \frac{27}{9} = 3 \leq \frac{work}{span} = \frac{T_1}{T_\infty} = \frac{27}{4} = 6.75 = Scalability_\infty \tag{4.26}$$

As explained in Section 2.5.4, the work-span model gives a more realistic upper bound of scalability as it takes into consideration that the parallel tasks *are not infinitely divisible*. That is, the red squares cannot be further divided. Nevertheless, the work-span model is harder to compute in realistic situations as the actual amount of work within each task needs to be known, as well as the way of splitting the work over threads.

4.8.1 Localization techniques

It is important to investigate how the localization techniques of Section 4.6 would affect the performance of the parallel algorithm. These techniques invalidate the assumption that all tasks have the same computation cost. This is sketched in Fig. 4.15. First, due to the infrequent and asynchronous updates of the sub-domain local matrices, some tasks in *BLOCK A* take less time to be performed and are noted with circles. Next, in *BLOCK C*, some injectors are latent, thus the linear equivalent model (4.14) is used and the corresponding tasks are

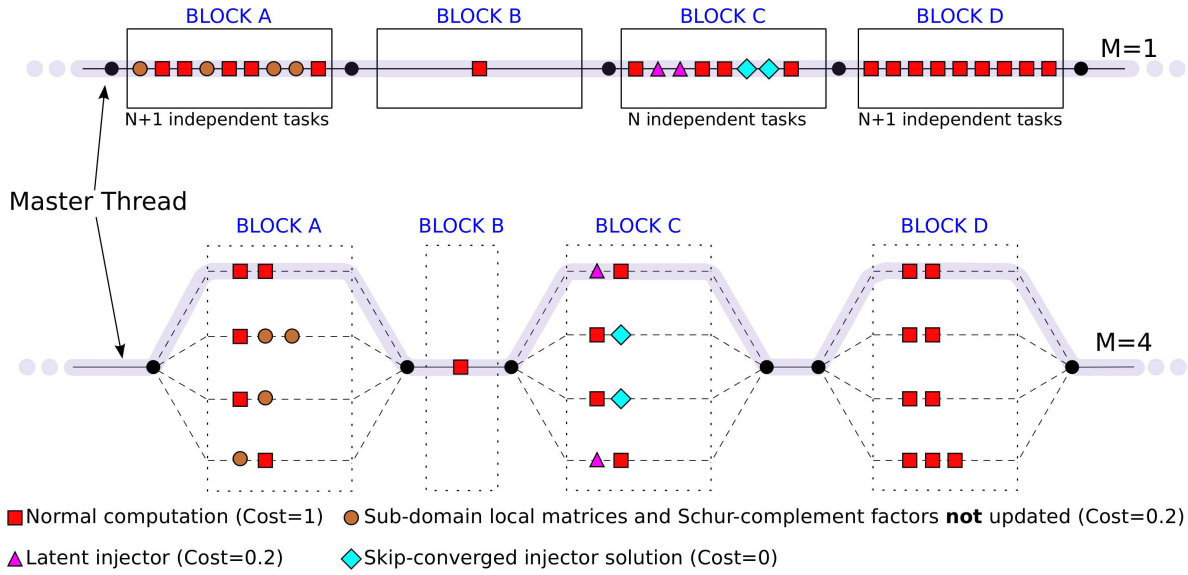


Figure 4.15: Variant of Fig. 4.14 with localization techniques

noted with triangles. Finally, some injectors have already converged in this iteration, thus they are noted in rhombus and are skipped. The fictional computation costs assumed are given in Fig. 4.15.

Therefore, using the work-span model to recompute scalability gives:

$$Scalability_4 = \frac{T_1}{T_4} = \frac{20.2}{7.2} = 2.8 \leq \frac{work}{span} = \frac{T_1}{T_\infty} = \frac{20.2}{4} = 5.05 = Scalability_\infty \tag{4.27}$$

where it can be seen that both the execution on one thread (T_1) and on four threads (T_4) are accelerated. Again, for this calculation the assumptions of ideal load balancing and no OHC were used.

Nevertheless, while the localization techniques decrease the overall simulation time ($T_1 = 20.2 < 27$ and $T_4 = 7.2 < 9$), the scalability of the algorithm is also decreased. The reason behind this is that the work (T_1) decreases immediately even with one reduced task (skip-converged, latent, etc.), while the span is harder to decrease. Taking the upper limit, T_∞ will only decrease if *all* tasks are reduced. That is, *if only a single task per parallel BLOCK remains* unreduced, T_∞ will be the same as if no task was reduced (in our case $T_\infty = 4$).

This observation introduces a trade-off between scalability and using localization techniques to accelerate the simulation. However, as long as T_M is decreasing, the speedup which is calculated against T_1^* (the run-time of the program with one worker using the fastest -or a very fast -sequential algorithm), is increasing. This complex behavior will be further examined in Section 4.9 using time measurements from the test systems.

4.8.2 Load balancing

In this subsection, the assumption of ideal load balancing considered above will be examined. First, it is obvious that all the tasks cannot have the same computational burden. For in-

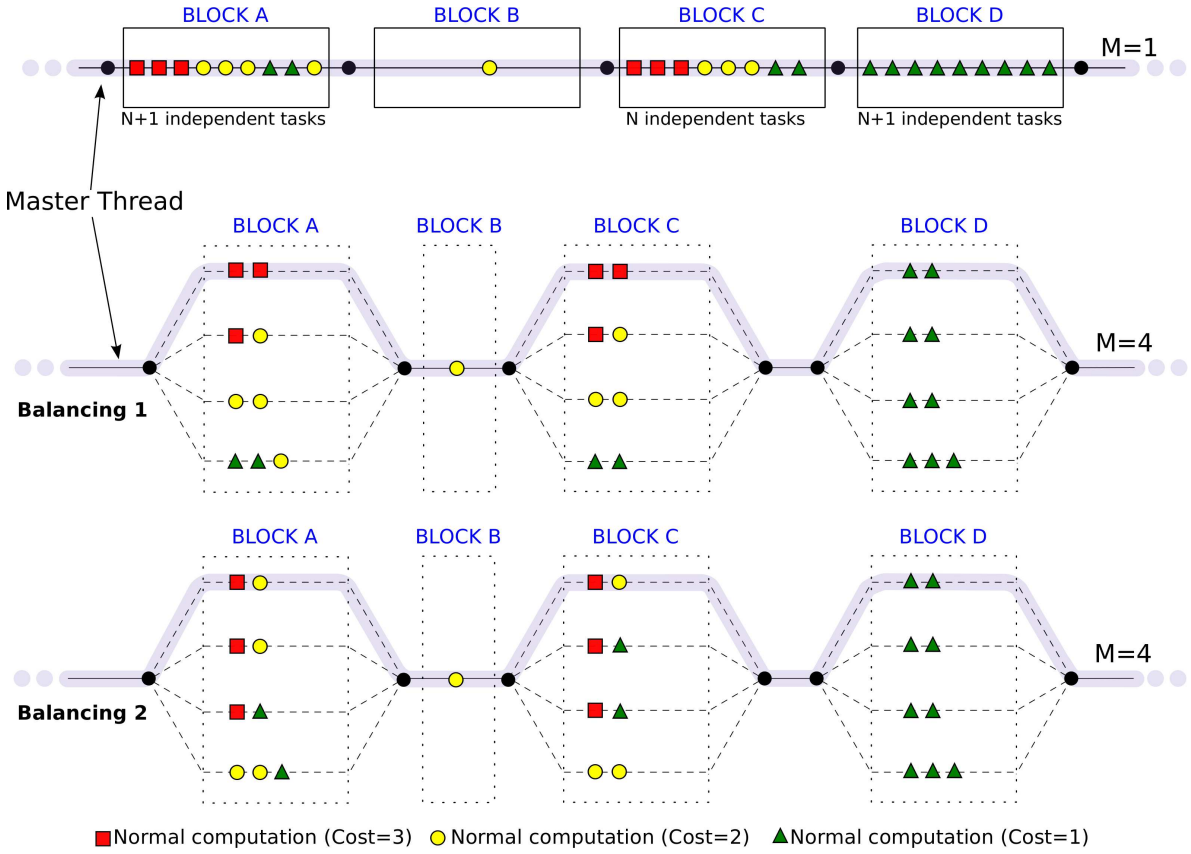


Figure 4.16: Variant of Fig. 4.14 with three different types of injectors

stance, *BLOCK A* involves the computation, discretization and factorization of sub-systems, which is in general more time consuming than a system solution performed in *BLOCK C*. Moreover, even within the same BLOCK, depending on the type and size of the injector model, the effort can vary. For example, the solution of voltage-sensitive exponential load model, that includes two algebraic equations, is much lighter than the solution of a type-3 wind turbine model involving more than 50 DAEs. Finally, the localization techniques also modify the cost of each task leading to even higher variation.

Let us consider another modified version of Fig. 4.14, with three different types of equipment, each with different computing cost, as shown in Fig. 4.16. With these costs, scalability can be computed for the two balancing schemes shown. In the first one, the tasks were naively split among threads according to their order in sequential execution ($M = 1$). Thus, with the work-span model:

$$Scalability_4 = \frac{T_1}{T_4} = \frac{47}{17} \simeq 2.8 \leq \frac{work}{span} = \frac{T_1}{T_\infty} = \frac{47}{9} \cong 5.2 = Scalability_\infty \tag{4.28}$$

In the second balancing, the tasks are split among threads so as to decrease T_4, that is the balancing mechanism is aware of the individual cost of each task. This leads to an improved

scalability:

$$Scalability_4 = \frac{T_1}{T_4} = \frac{47}{15} \simeq 3.1 \leq \frac{work}{span} = \frac{T_1}{T_\infty} = \frac{47}{9} \cong 5.2 = Scalability_\infty \tag{4.29}$$

Of course the maximum scalability is exactly the same as it is dictated by the most computationally costing task of each BLOCK, and this does not change due to different balancing strategies.

In the test systems considered in Section 4.9, the size of injector models varies from 2 to 60 variables. This means that the computational tasks will also vary significantly. If the tasks are not split over the threads properly, scalability will suffer. However, building a task-size-aware balancing strategy is difficult and would need a lot of bookkeeping and rearranging of tasks from one parallel segment to the next. In addition, the localization techniques modify the cost of each task from iteration to iteration, depending on the dynamic behavior of the system (which is unknown beforehand), thus making the task size less predictable.

In such situations, the *dynamic* balancing strategy of OpenMP can offer the desirable load balancing results (see Section 2.6.3). With this strategy, a set of *consecutive* tasks called *chunk* is given to each thread. When a thread has completed its assigned tasks, a new chunk of tasks is given to it. For example, if the dynamic strategy with a single task per chunk is used in Fig. 4.16, the best result of *"Balancing 2"* is obtained. Although the small chunk selected guarantees a close to optimal balancing, it also means that the threads need to frequently return and ask for more work, which translates to increased OHC. Moreover, spatial locality (see Section 2.6.2) suffers as the tasks assigned to each thread are "far" from each other, thus their data in memory are probably "far" as well.

These problems can be addressed by defining the chunk to be larger than one. This way, the threads do not return as often to ask for more work, and by treating several consecutive tasks it is more likely that their data are also consecutive in memory (this is the case in RAMSES). Larger chunks decrease the OHC but can cause imbalance among threads as some chunks might contain more computationally expensive tasks than others. Usually, a compromise is made with a chunk small enough to achieve good load balancing but adequately large to decrease the OHC and exploit spatial locality. In the simulations of Section 4.9, a default size of chunk equal to $\max\left(\left\lceil \frac{N}{4M} \right\rceil, 1\right)$ was found to be satisfactory through a trial-and-error procedure.

While the dynamic scheduling strategy of OpenMP can address load balancing and spatial locality, temporal locality (see Section 2.6.2) cannot be easily achieved. The reason is that the tasks treated by each thread, and thus the data accessed, are decided at run-time and can change from one parallel segment of the code to the next. When executing on UMA architecture computers, where access time to a memory location is independent of which processor makes the request, the lack of temporal locality is not crucial.

On the other hand, when executing on NUMA architecture computers, the lack of temporal locality can introduce a high OHC. In this case, the *static* scheduling strategy of OpenMP

Table 4.1: Balancing strategies used in RAMSES

Computer architecture	Scheduling	Bind thread to CPU	Chunk
NUMA (1 of Section 2.7)	Static	yes	$\max\left(\left\lceil \frac{N}{4M} \right\rceil, 1\right)$
UMA (2 and 3 of Section 2.7)	Dynamic	yes	$\max\left(\left\lceil \frac{N}{4M} \right\rceil, 1\right)$

can be more effective. With this strategy, each thread always handles the *same* sub-domains and accesses the *same* data at each parallel segment, thus increasing temporal locality. Of course, chunks are also employed to increase spatial locality and to randomize the distribution of sub-domains to threads, thus decreasing the possibility that all injectors of the same type are assigned to the same thread.

Summarizing, in the results of Section 4.9, two different balancing strategies have been used depending on the computing machine used, as shown in Table 4.1.

In both cases, the first touch memory allocation strategy has been used and each computational thread is bound to a physical core, as explained in Section 2.6.2. These mechanisms minimize the transfer of data among different cores, which can lead to increased OHC.

4.8.3 Overhead cost

The OHC considered in this work is associated to making the code run in parallel, managing the threads and the communication between them. In the previous sub-sections, it was shown that the OHC can be increased by dynamic load balancing (which increases the effort needed to manage the threads) and the lack of locality (which initiates data transfers and increases the communication).

However, even if these causes are ignored, there is still OHC relating to the creation and management of the *thread pool*[1] at the *fork* points and the synchronization at the *join* points in the code. Some implementations of the OpenMP library (like the one provided by Intel and used in this work) allow keeping the thread pool alive in the background, during the entire simulation, to decrease the cost. Nevertheless, the OHC is non-negligible and is dependent on the number of threads, the operating system, and the computer used.

As long as the parallel work of each thread is still divisible (that is, it consists of more than one task) and we assume "good" load balancing, Eq. 2.5 of Amdahl's law can be used to calculate the scalability without OHC. This can be then compared with the real scalability calculated with Eq. 2.1 and the difference between them is the OHC, given by:

$$OHC\,(M) = \frac{T_S + T_P}{T_S + \frac{T_P}{M}} - \frac{T_1}{T_M} \tag{4.30}$$

where the values of T_S and T_P can be acquired through a profiling of the algorithm in sequential execution ($M = 1$).

[1] Group of threads used to compute the tasks.

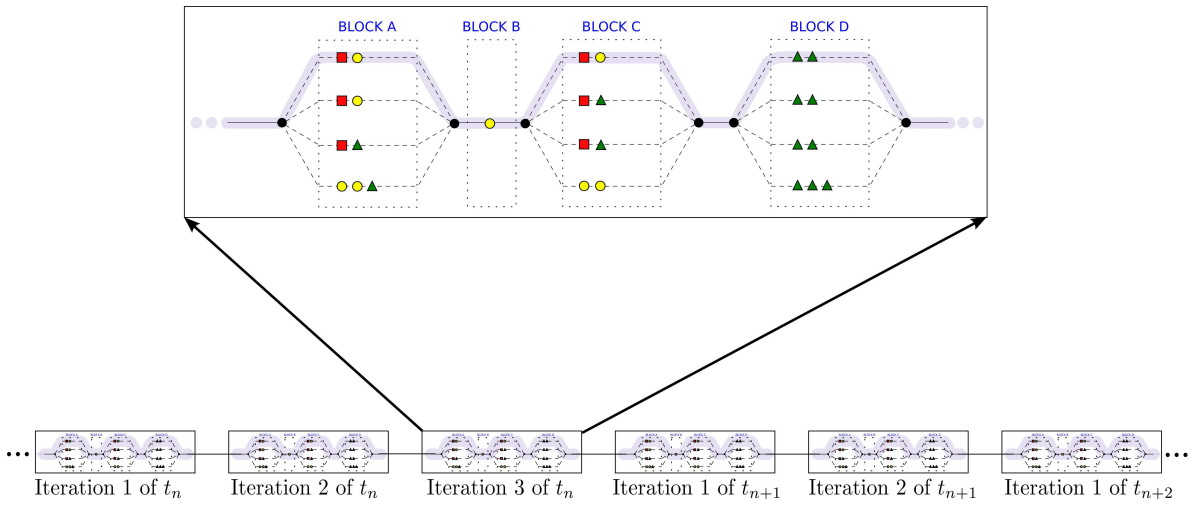


Figure 4.17: Several consecutive iterations of the algorithm in Fig. 4.3

Finally, when executing on NUMA computers, such as Machine 1 (see Table 4.1), the OHC varies depending on the location of the computational threads within the computer sockets. For example, Machine 1 consists of *four* identical sockets, each hosting *two* NUMA nodes with *six* cores (as sketched in Fig. 2.4). Thus, when using six cores assigned to one NUMA node, the OHC is small as the communication is faster. When using $6 - 12$ threads assigned to a single socket, the OHC is slightly larger. Finally, when threads are assigned to different sockets, the OHC increases depending on the communication speed between sockets.

4.8.4 Profiling

In this work, two types of profiling are used: numerical and time. Both profilings consider the entire simulation and not individual iterations of the algorithm. For example, the operations sketched in Fig. 4.16 relate to only *one* out of the thousands of iterations needed to perform a whole dynamic simulation, while Fig. 4.17 shows several such iterations over three successive time steps (t_n, t_{n+1}, t_{n+2}). Considering each individual iteration in the profiling would be impossible. Thus, a sum of the same operations over the different iterations is used.

For the numerical profiling, the total number of numerical operations relating to the execution of the algorithm are considered. For instance, the total numbers of sub-domain matrix factorizations performed at *BLOCK A*, of solutions of the reduced system (4.12), etc. The number of operations does not change when parallelizing them; it changes however when localization techniques are used. A selection of numerical profilings is shown in Appendix D.

For the time profiling, the time spent in each block of Fig. 4.3 in all iterations is presented as a percentage of the total time. The profiling is performed using a sequential execution of the algorithm ($M = 1$). In addition, there are some other operations shown in the "*Remaining BLOCK*" of Fig. 4.3. These operations relate to updating latency criteria, bookkeeping, treat-

Table 4.2: Options considered in the simulations

Configuration	Skip converged injectors	Update sub-domain matrices asynchronously	Exploit latency
I	✗	✗	✗
II	✓	✓	✗
III	✓	✓	✓

ing the discrete events, computing the actions of Discrete ConTroLlers (DCTL), etc. (more details about these operations are given in Appendix C). Some of them are performed in parallel and others not, thus time profiling these *"Remaining"* operations is also needed to correctly calculate the sequential and parallel times T_s and T_P. A selection of time profilings is presented alongside the results in the next section.

4.9 Experimental results

In this section some experimental results are given, using the test systems summarized in Section 1.3. First, the contingency considered in each test system will be described. Next, several executions of the same simulation will be performed using a combination of simulation parameters. These are shown in Table 4.2.

In addition, the non-decomposed approach with the use of an integrated VDHN scheme, described in Section 1.2.5.2, will be used as a benchmark. That is, it will be used to assess the accuracy of the dynamic response and to provide T_1^* in Eq. 2.2, for the speedup calculation. For simplicity, this benchmark algorithm will be referred to as *integrated* hereon.

Both the proposed algorithm and the integrated are implemented in RAMSES. The same models, convergence tolerance, algebraization method (second-order BDF), and way of handling the discrete events are used. For the solution of the sparse systems (the integrated Jacobian or the reduced system of Eq. 4.12), the sparse linear solver HSL MA41 [HSL14] is used. For the solution of the much smaller, dense injector linear systems (4.6), Intel MKL LAPACK library is used. The matrix update criteria are as follows: for the integrated and Config. I, all the matrices are updated every five iterations until convergence; for Configs. II and III, the matrices of *each* sub-domain are updated every five iterations unless it has already converged. Finally, the convergence checks defined in Eqs. 1.16a and 1.17 are used, with $\epsilon_g = \epsilon_{frel} = \epsilon_{fabs} = 10^{-4}$. Keeping the aforementioned parameters and solvers of the simulation constant for both algorithms permits a (more) rigorous evaluation of the proposed algorithm performance.

The main investigations will be performed on Machine 1 (see Section 2.7), but a performance comparison with Machines 2 and 3 will be shown in Section 4.9.4.4.

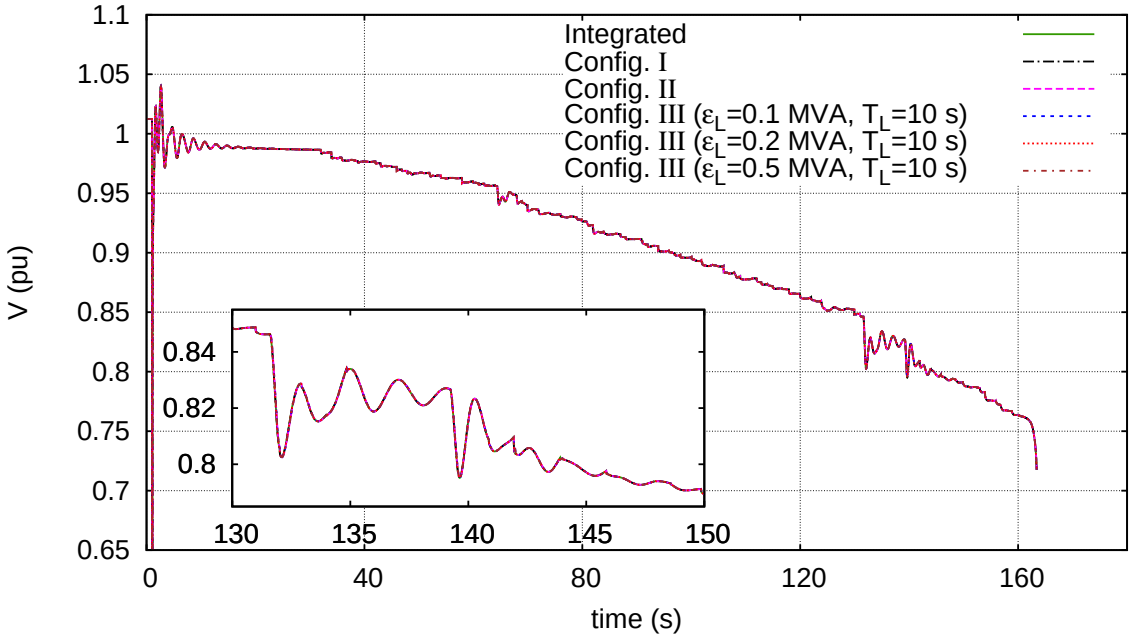


Figure 4.18: Nordic (operating point A): Voltage evolution at bus 1041

4.9.1 Nordic system

This is the smallest test system considered in this work with 750 DAEs. When the aforementioned decomposition is applied, the network sub-domain is formed including 77 buses with 154 variables and 43 injector sub-domains with the remaining variables. While the proposed algorithm is designed to tackle large-scale systems, it will be shown that even small systems such as this one can benefit by the algorithm, to a smaller degree. Moreover, this small system has been frequently used for the validation of the proposed simulation methods and localization techniques as its small size makes it easier to detect problems and verify the dynamic response of the system.

For this system, two different operating points were considered: an N-1 insecure point A and a secure point B. More details on these operating points can be found in [VP13]. The disturbance of concern is a three-phase solid fault on line $4032 - 4044$, near bus 4032, lasting five cycles (i.e. 100 ms) and cleared by opening the line, which remains opened. Next, the system is simulated over an interval of 240 s with a time-step size of one cycle. It evolves in the long term under the effect of 22 automatic Load Tap Changers (LTCs) trying to restore the distribution voltages as well as the synchronous generator OvereXcitation Limiters (OXLs).

4.9.1.1 Operating point A

The evolution of the transmission system voltage at bus 1041 is shown in Fig. 4.18, the rotor speed of generator $g15$ in Fig. 4.19, and the apparent power output of the same generator in Fig. 4.20. The curves have been computed using the *Integrated* method as well as the

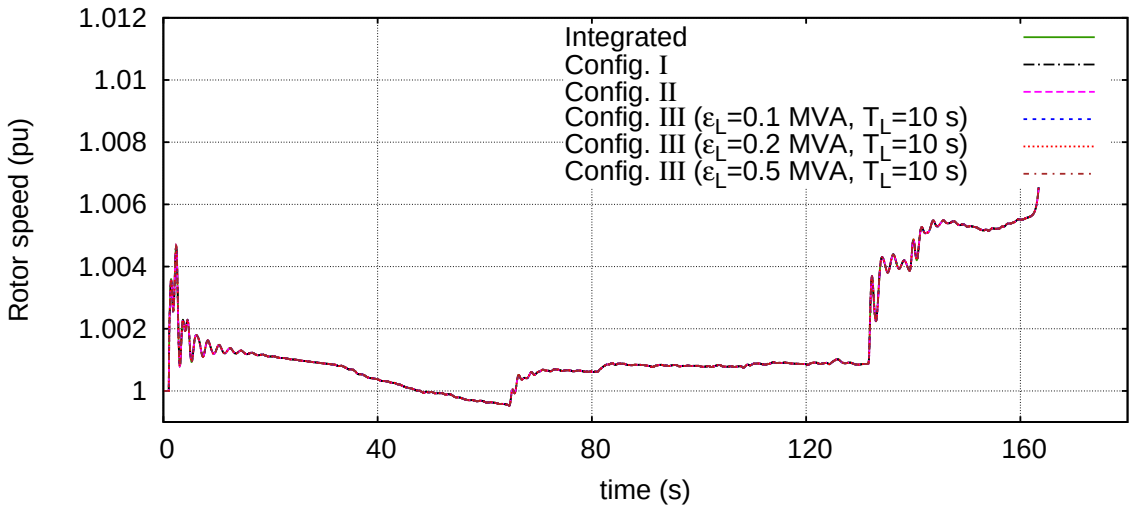


Figure 4.19: Nordic (operating point A): Evolution of rotor speed of generator $g15$

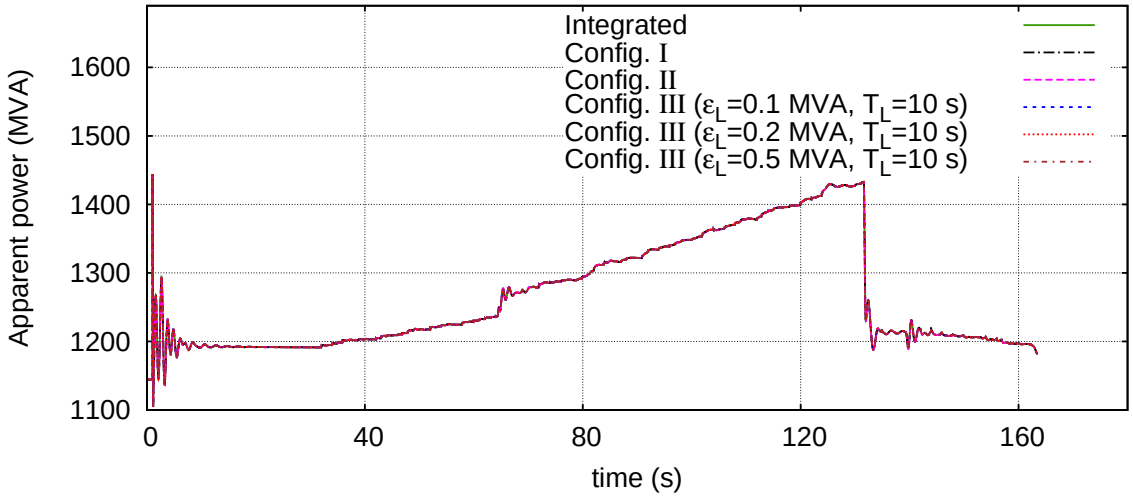


Figure 4.20: Nordic (operating point A): Apparent power output of generator $g15$

configurations listed in Table 4.2. In response to the initial disturbance, the system undergoes electromechanical oscillations that die out in 20 seconds. Then, the system settles at a short-term equilibrium, until the LTCs start acting at $t = 35$ s. Subsequently, the voltages evolve under the effect of LTCs and OXLs. The system is long-term voltage unstable and eventually collapses less than three minutes after the initiating line outage. The dynamic behavior is thoroughly examined in [VP13].

Table 4.3 shows the simulation time, the speedup (computed using Eq. 2.2), and the maximum inaccuracy over all the bus voltages, compared to the integrated method. From the sequential execution timings, it can be seen that Config. I performs better than the integrated, even though the two perform the same number of iterations, matrix updates, etc. Understanding this performance difference is challenging. First, the decomposed algorithm

Table 4.3: Nordic (operating point A): Execution times and inaccuracy in simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	5.8 / -	- / -	-
Config. I	3.7 / 1.6	2.3 / 2.5	0
Config. II	3.7 / 1.6	2.3 / 2.5	0
Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)	3.9 / 1.5	2.6 / 2.2	0
Config. III ($\epsilon_L = 0.2$ MVA, $T_L = 10$ s)	3.9 / 1.5	2.6 / 2.2	0
Config. III ($\epsilon_L = 0.5$ MVA, $T_L = 10$ s)	3.9 / 1.5	2.6 / 2.2	0

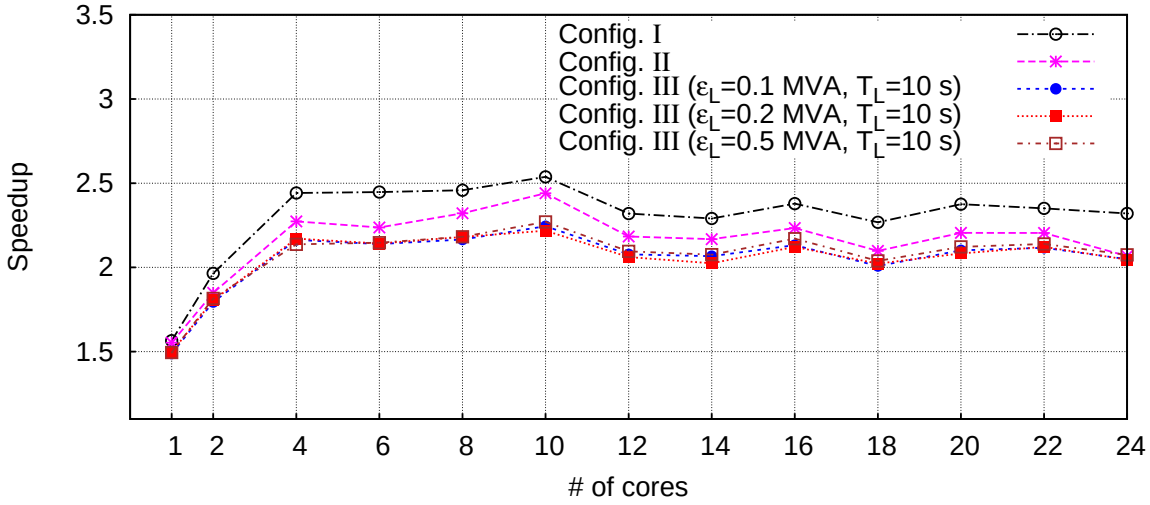


Figure 4.21: Nordic (operating point A): Speedup computed with Eq. 2.2

has extra OHC related to bookkeeping, which penalizes its performance. Second, the integrated method solves a linear system of size 750 (shown in Eq. 4.20) at every iteration using the sparse solver; while, for the same iteration the decomposed algorithm solves a linear system of size 154 (two times the number of buses) *and* 43 smaller, dense linear systems using the Lapack LU procedures. While the number of linear matrix updates, factorizations, and system solutions are the same, the two approaches have different execution times, with the latter being in most of the systems tested faster. Thus, the performance difference between the two simulations (integrated and Config. I) is a combination of these two factors and varies from system to system and even from contingency to contingency. However, the speedup of Config. I in all the test systems and contingencies checked ranges between 0.9 and 1.3 of the integrated.

Configurations II and III do not offer higher performance. This is due to the nature of the test case: a collapsing scenario, where the entire system is driven to instability does not

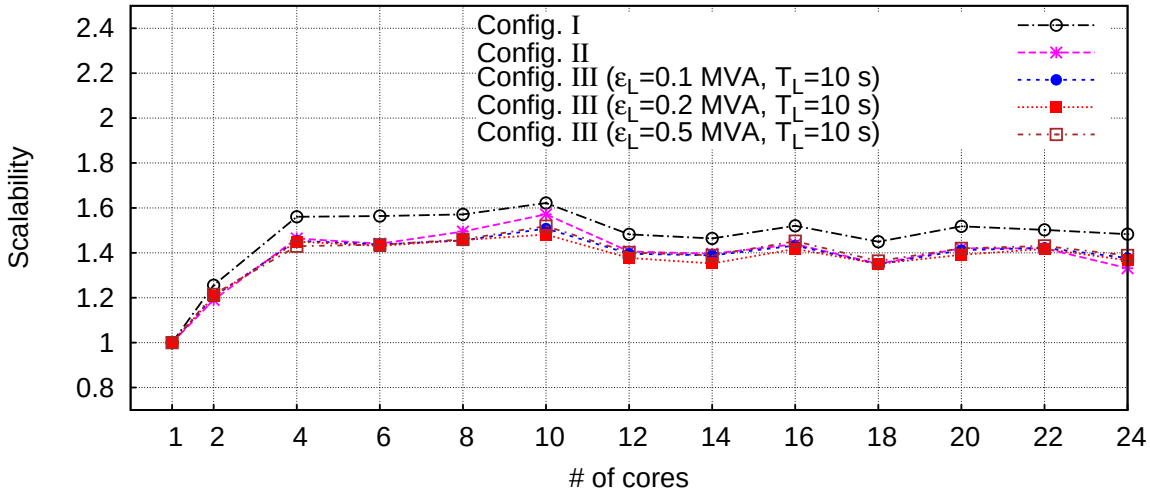


Figure 4.22: Nordic (operating point A): Effective scalability computed with Eq. 2.1

allow to exploit localization. In addition, Config. III has the extra burden of computing the EMA and standard deviation of each injector for the switching Algorithm 4.1. The sequential performance of all systems will be further discussed in Section 4.9.4.1.

The scalability and speedup for a varying number of computational threads (up to 24) are shown in Figs. 4.21 and 4.22, respectively. It can be seen that the efficiency of the parallelization (see Section 2.5.1) is maximum at $M = 4$ while the best results are acquired at $M = 10$ (these are the ones shown at the parallel execution column of Table 4.3). When using more than ten computational threads, the OHC of creating and managing the extra threads overtakes the incremental gain, thus the speedup declines. In general, for such reduced in size systems, small UMA computers of up to four cores provide the best output in terms of efficiency of parallelism.

Finally, it can be seen from Figs. 4.18-4.20 that all six curves are indiscernible. As explained in Section 4.7, it is expected that Configs. I and II give exactly the same output as the Integrated. On the other hand, Config. III is expected to introduce some inaccuracy to the simulation, especially for larger latency tolerance (ϵ_L). However, as this is a collapsing scenario with large dynamic activity, *none of the injectors gets latent* and the system response is exactly the same as with the other configurations.

4.9.1.2 Operating point B

Similarly, the evolution of the transmission system voltage at bus 1041 is shown in Fig. 4.23 and the rotor speed of generator $g15A$ in Fig. 4.24. Contrary to Operating point A, the response at Operating point B is long-term stable. After the electromechanical oscillations have died out, the system evolves in the long term under the effect of LTC devices acting to restore distribution voltages. Thus, the decision about the stability of the system can only be made after the simulation of the entire time horizon.

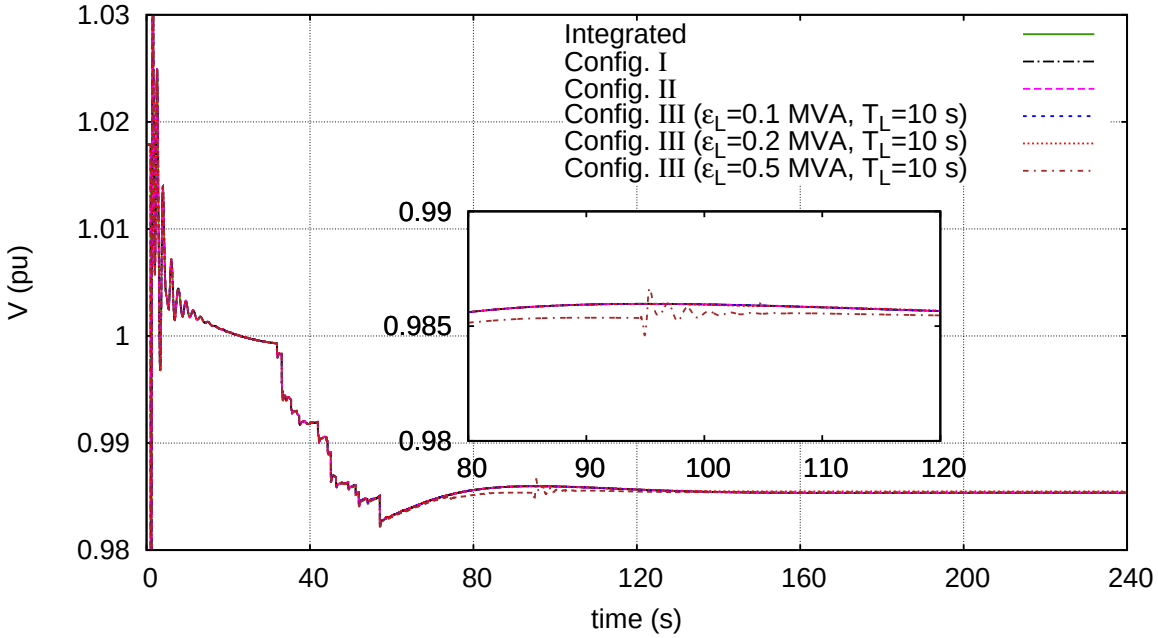


Figure 4.23: Nordic (operating point B): Voltage evolution at bus 1041

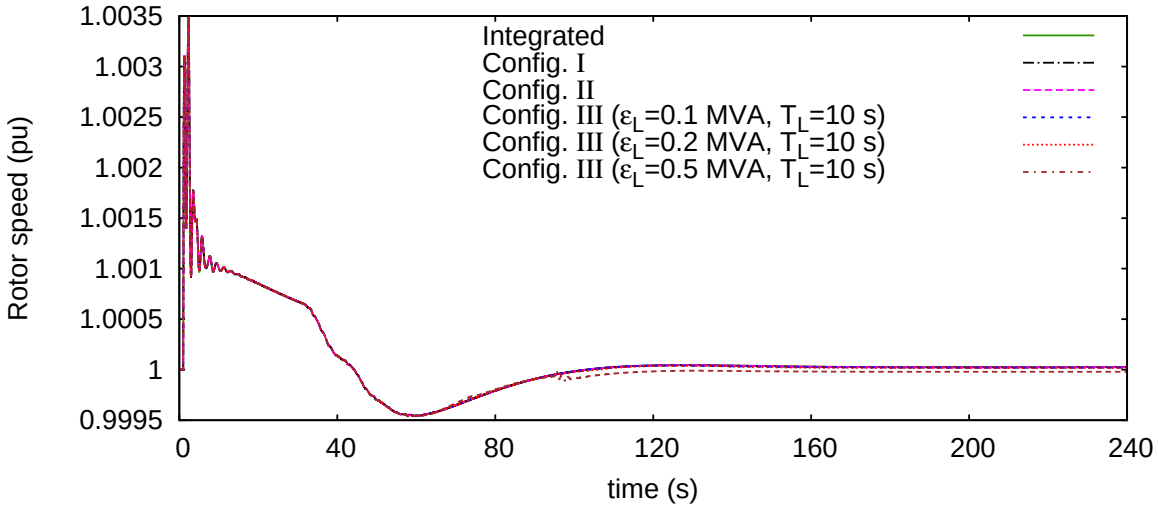


Figure 4.24: Nordic (operating point B): Evolution of rotor speed of generator $g15A$

Table 4.4 shows the simulation time, the speedup and the maximum inaccuracy over all the bus voltages compared to the integrated method. From the sequential execution timings, it can be seen that Config. I performs better than the integrated. For this -non collapsing- scenario, the localization techniques also offer significant speedup leading up to 1.7 faster execution compared to the integrated.

Figure 4.25 shows the absolute error on the voltage of transmission bus 1041, i.e. the absolute difference between the value computed with the Integrated and with Config. III. By simulating several scenarios with different latency tolerance values, it was found that an

Table 4.4: Nordic (operating point B): Execution times and inaccuracy in simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	4.0 / -	- / -	-
Config. I	3.6 /1.1	2.5 / 1.6	0
Config. II	3.3 / 1.2	2.4 / 1.7	0
Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)	2.6 / 1.5	2.3 / 1.7	0.0006
Config. III ($\epsilon_L = 0.2$ MVA, $T_L = 10$ s)	2.4 / 1.7	2.3 / 1.7	0.0015
Config. III ($\epsilon_L = 0.5$ MVA, $T_L = 10$ s)	2.3 / 1.7	2.2 / 1.8	0.0020

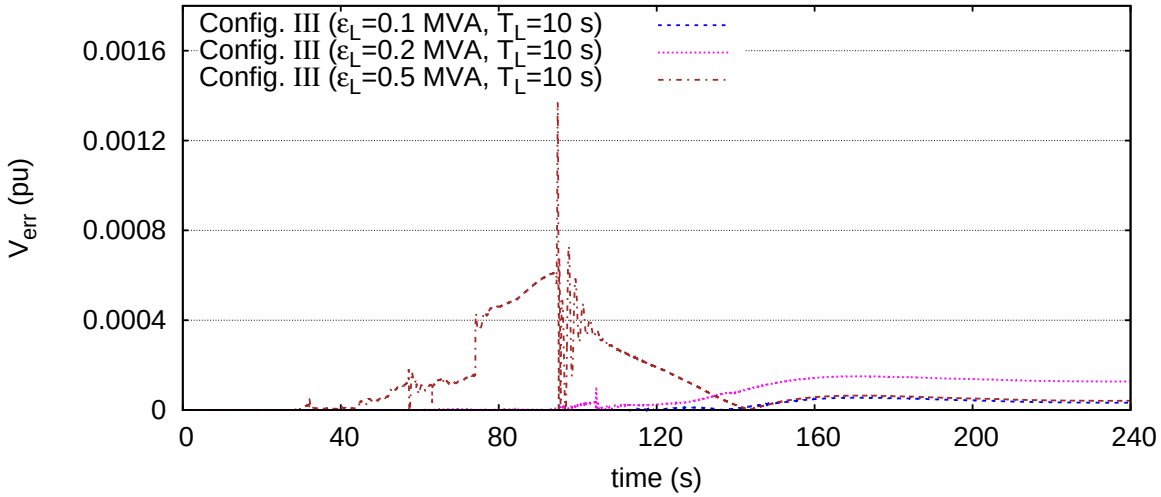


Figure 4.25: Nordic (operating point B): Absolute voltage error on bus 1041

$\epsilon_L \leq 0.2$ MVA gives a good speedup while keeping the inaccuracy introduced at minimum.

Figure 4.26 shows the apparent power output of generator $g15A$ computed with Config. III for three different values of ϵ_L. In Config. III, the power plant is identified as latent (i.e. its full model is replaced by the sensitivity model) in the time intervals shown in gray and active in the rest. The vertical black lines show the transitions between modes. Larger values of ϵ_L lead to earlier switching of injectors to latent. However, when latency is used, the increased speedup comes with some inaccuracy introduced to the simulation results.

From the parallel execution timings of Table 4.4, it can be seen that Configs. I and II gain when executed in parallel. On the other hand, the simulations with Config. III do not gain further between the sequential and parallel execution. This is due to the increased number of injectors becoming latent thus decreasing the amount of parallel work (see Section 4.8.1).

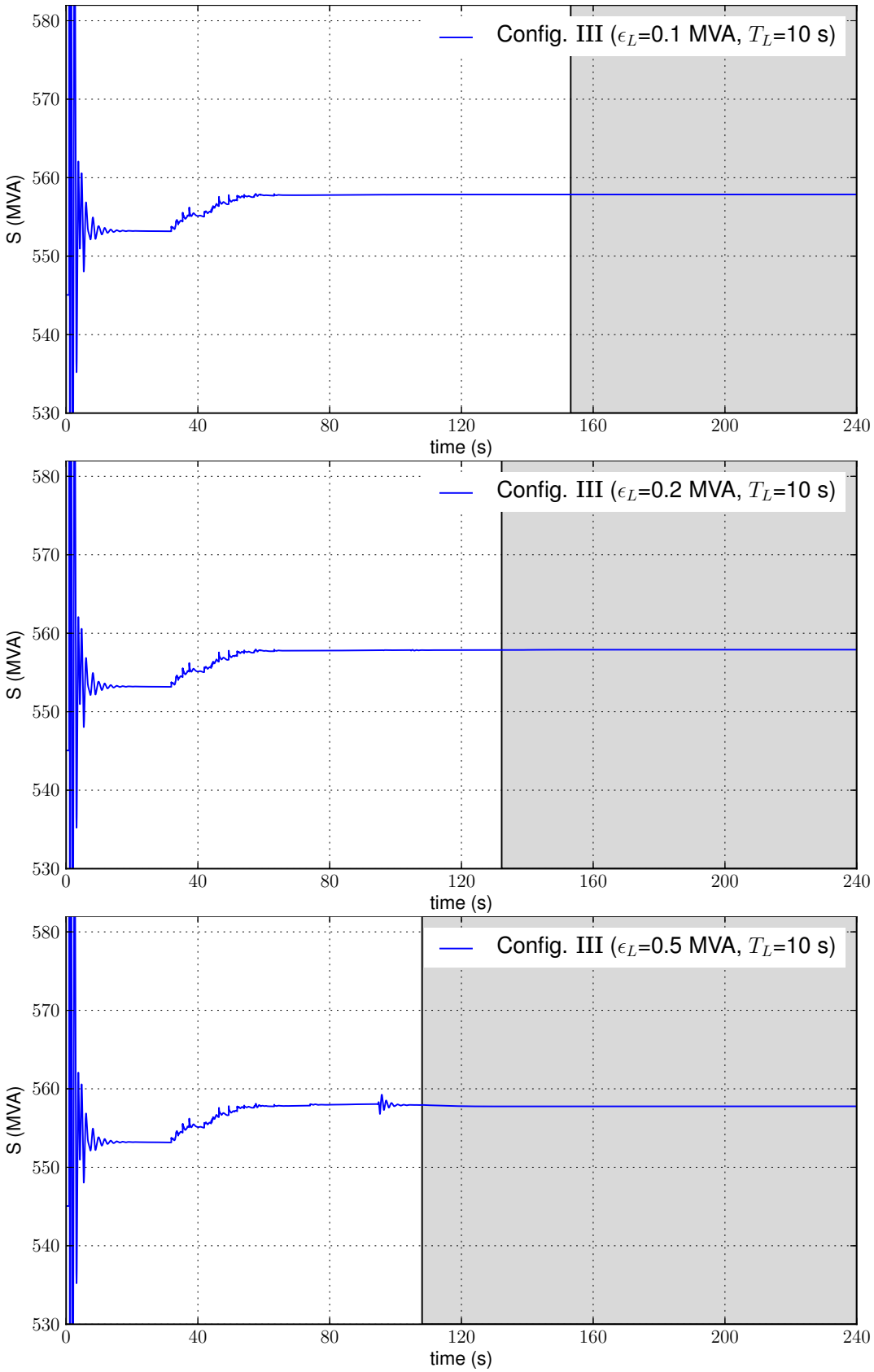


Figure 4.26: Nordic (operating point B): Apparent power output of generator $g15A$

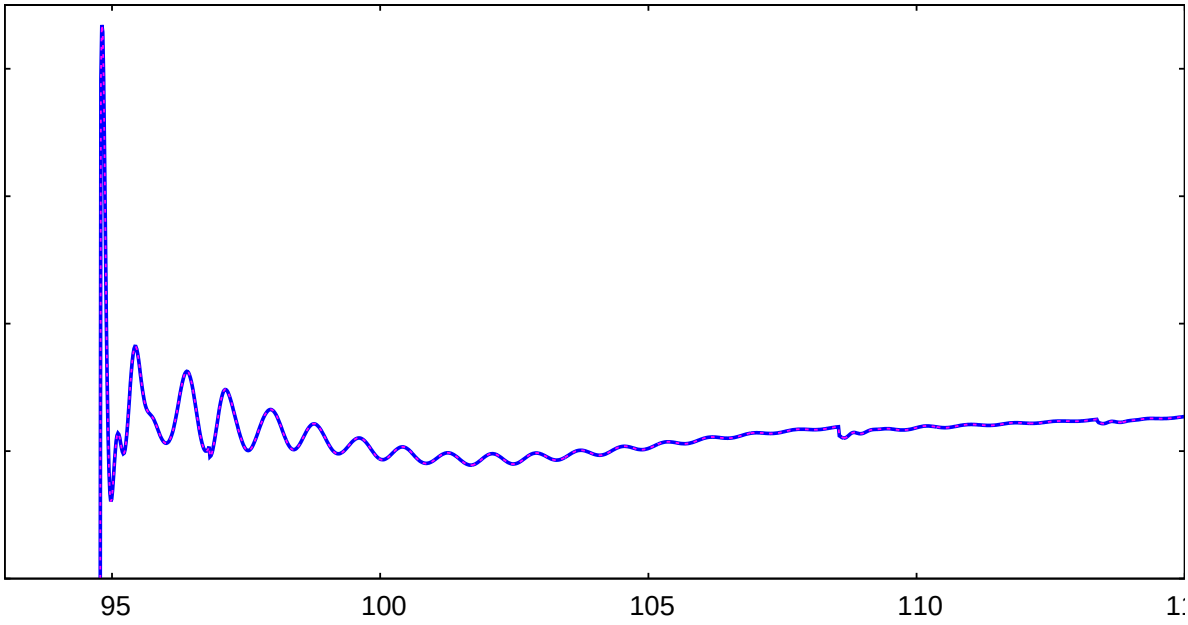


Figure 4.27: HQ: Voltage evolution at bus 702

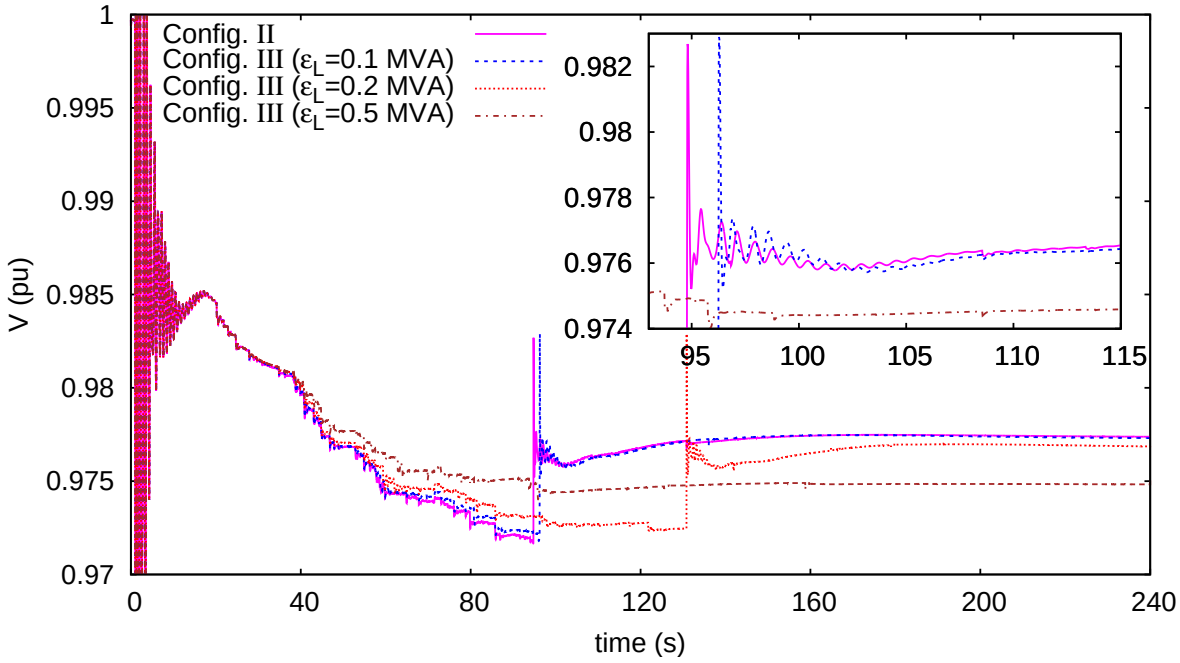


Figure 4.28: HQ: Voltage evolution at bus 702

4.9.2 Hydro-Québec system

This is the medium test system considered in this work with 35559 DAEs. The system is decomposed into the network sub-domain, including 2565 buses, and the $N = 4601$ injectors. The disturbance consists of a short circuit near bus 702 lasting six cycles (at 60 Hz), that is

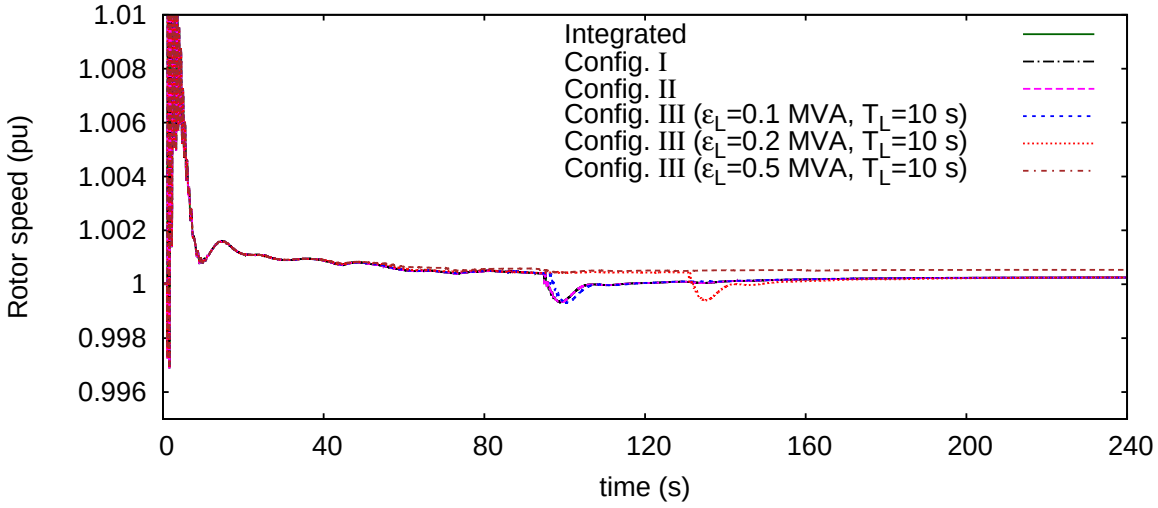


Figure 4.29: HQ: Evolution of rotor speed of generator 294

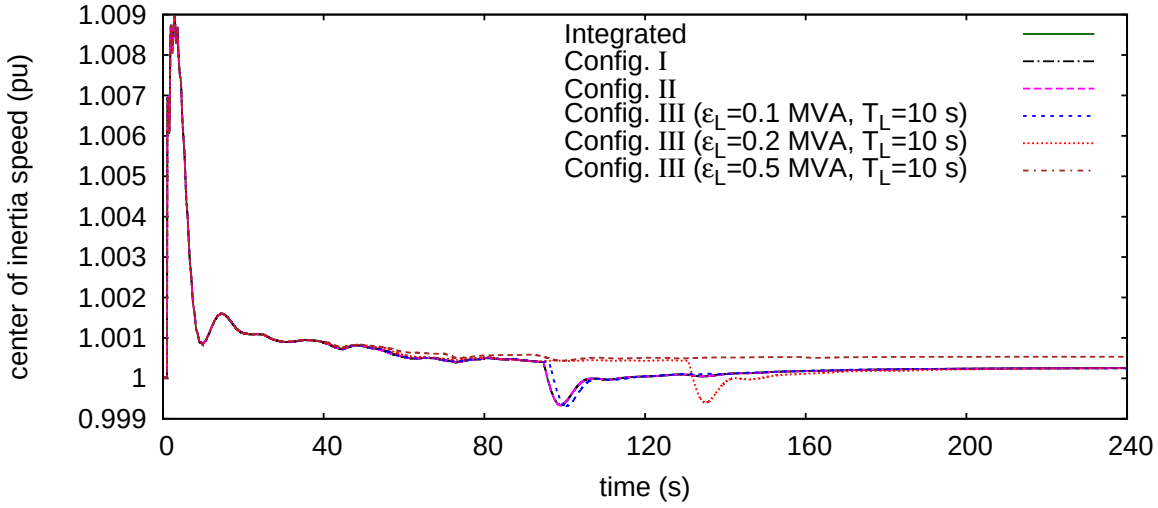


Figure 4.30: HQ: Center of inertia speed of the system

cleared by opening a 735-kV line. Then, the system is simulated over an interval of 240 s with a time-step of one cycle. It evolves in the long term under the effect of 1111 LTCs, 25 Automatic Shunt Reactor Tripping (ASRT) devices, as well as synchronous generator OXLs.

Figures 4.27 and 4.28 show the voltage evolution at the bus nearest to the fault using the Integrated method and the DDM-based with the three parameter configurations of Table 4.2. Similarly, Figs. 4.29 and 4.30 show the rotor speed of a generator and the system center of inertia speed, respectively. From all three figures, it can be seen that the Integrated method and the DDM-based with Configs. I and II give exactly the same response, as discussed in Section 4.7.

However, when latency is used (Config. III), it can be seen that the system response is modified and some inaccuracy is observed. More specifically, with $\epsilon_L = 0.1$ MVA, the

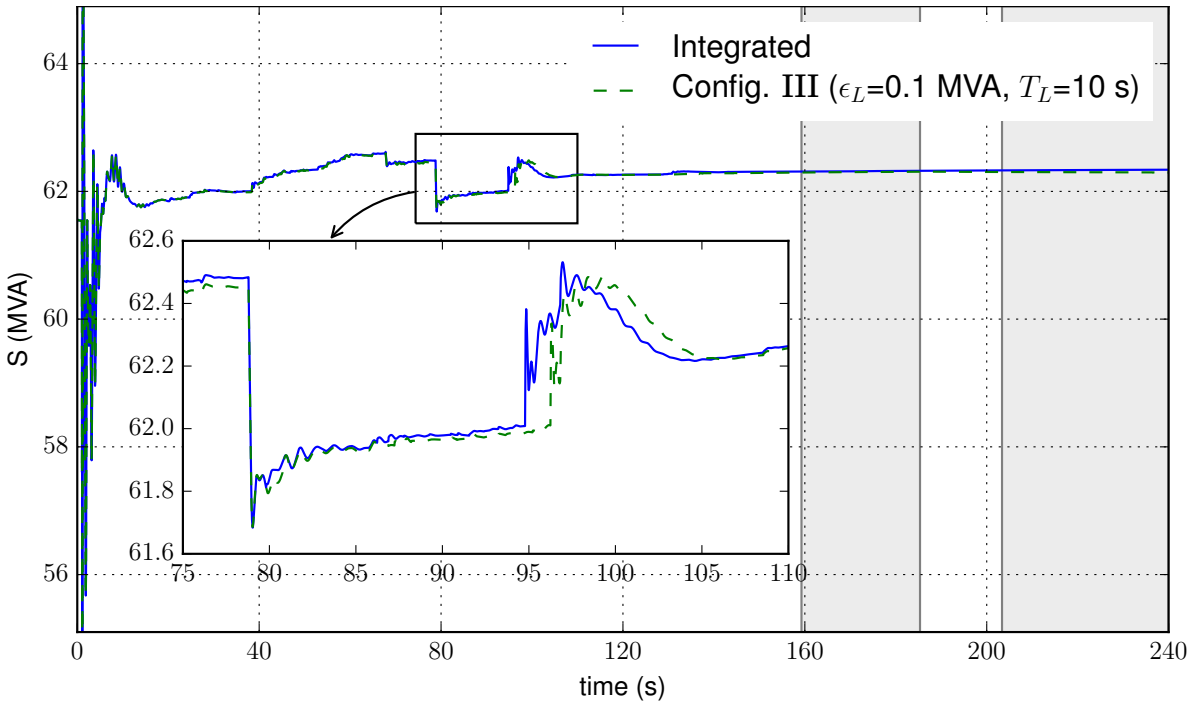


Figure 4.31: HQ: Apparent power output of synchronous generator 8517

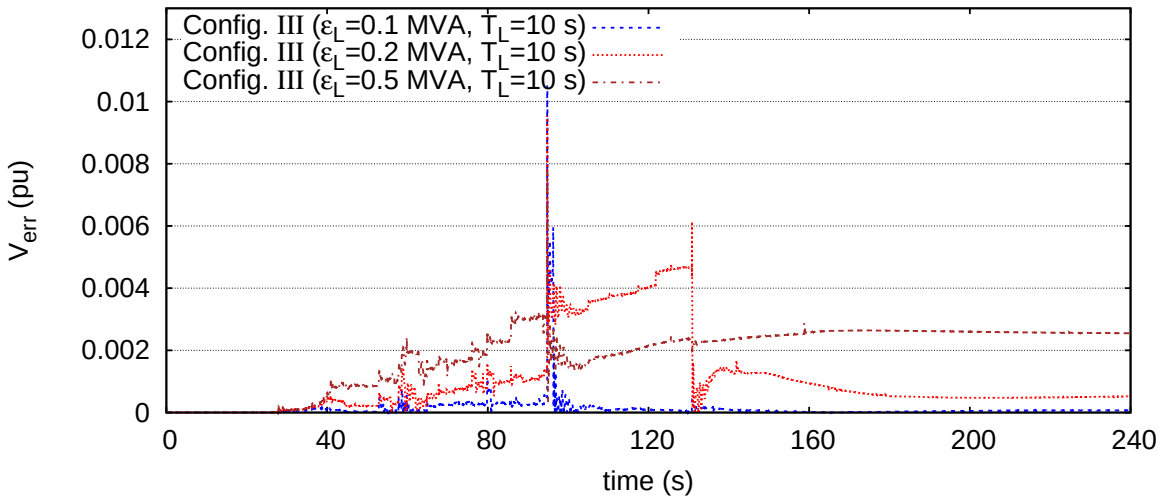


Figure 4.32: HQ: Absolute voltage error on bus 702

difference is almost indiscernible. With $\epsilon_L = 0.2$ MVA, the operation of the ASRT device (which depends on monitoring the voltage magnitude) is delayed from $t = 94$ s to $t = 131$ s. It must be noted that, although the difference seems large at first glance, it is considered acceptable by the Hydro-Québec (HQ) engineers. In fact, the voltage monitored by the ASRT device evolves marginally close to the triggering threshold, and a small difference in system trajectory is enough to postpone its action. However, the final transmission voltage reaches a value very close to the one obtained without approximation (bringing the final voltage profile

Table 4.5: HQ: Execution times and inaccuracy in simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	413.8 / -	- / -	-
Config. I	322.4 / 1.3	57.0 / 7.3	0.0
Config. II	260.7 / 1.6	47.0 / 8.8	0.0
Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)	125.9 / 3.3	40.8 / 10.1	0.010
Config. III ($\epsilon_L = 0.2$ MVA, $T_L = 10$ s)	107.3 / 3.9	37.4 / 11.1	0.012
Config. III ($\epsilon_L = 0.5$ MVA, $T_L = 10$ s)	92.3 / 4.5	35.5 / 11.7	0.020

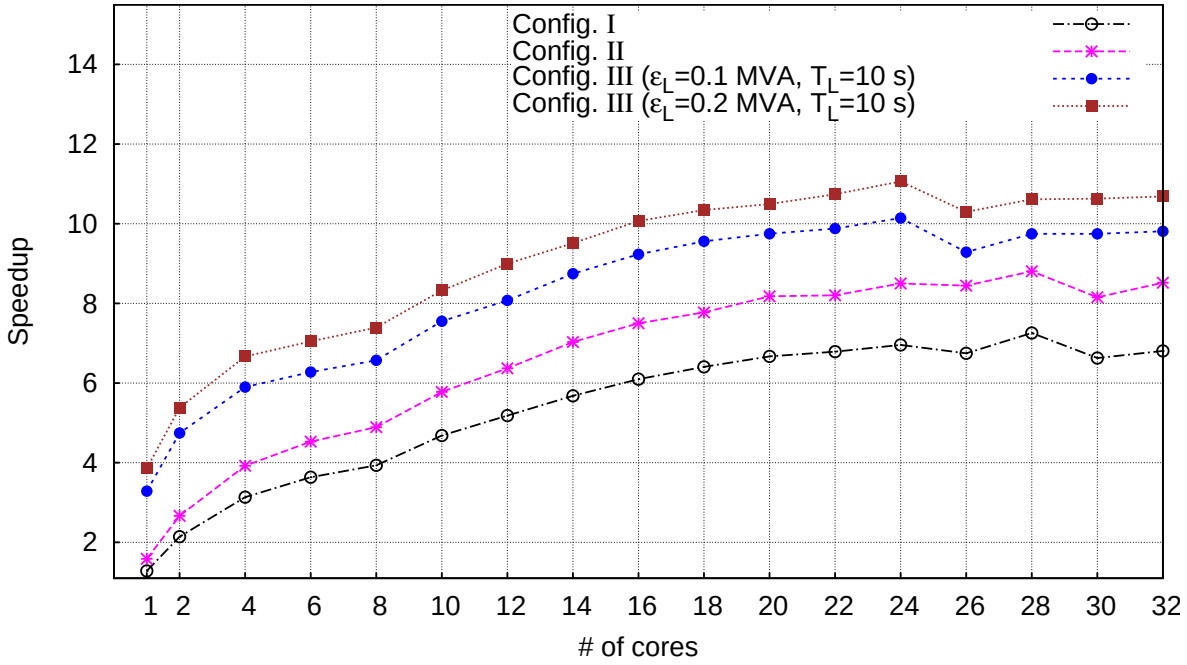


Figure 4.33: HQ: Speedup computed with Eq. 2.2

above some threshold values is the overall objective of the ASRT devices). The delayed response is observed on the frequency due to load sensitivity to voltage.

An even higher latency tolerance of $\epsilon_L = 0.5$ MVA, leads to the ASRT device not being triggered. Although this case is less satisfactory, a discrepancy of one on the number of tripped reactors is still considered acceptable by the HQ engineers. The reason for the ASRT not being triggered is the one exposed above.

Figure 4.31 shows the apparent power of the generator of a hydro power plant close to the fault location. It can be seen that the plant goes latent at $t = 159$ s, gets back to active mode once for a short period of time at $t = 185$ s and gets latent again at $t = 203$ s. Figure 4.32 shows the absolute voltage error of transmission bus 702. It can be seen that the error peak

Table 4.6: HQ: Time profiling of sequential execution ($M = 1$)

	% of execution time		
	Config. I	Config. II	Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)
BLOCK A	13.61	8.66	6.53
BLOCK B	4.14	2.2	4.66
BLOCK C	67.47	68.25	52.06
BLOCK D	3.11	4.88	8.34
Remaining parallel	4.31	3.95	6.21
Remaining sequential	7.36	11.06	22.2
T_P (100-T_S)	88.5	86.74	73.14
T_S (BLOCK B+Rem. Seq.)	11.5	13.26	26.86

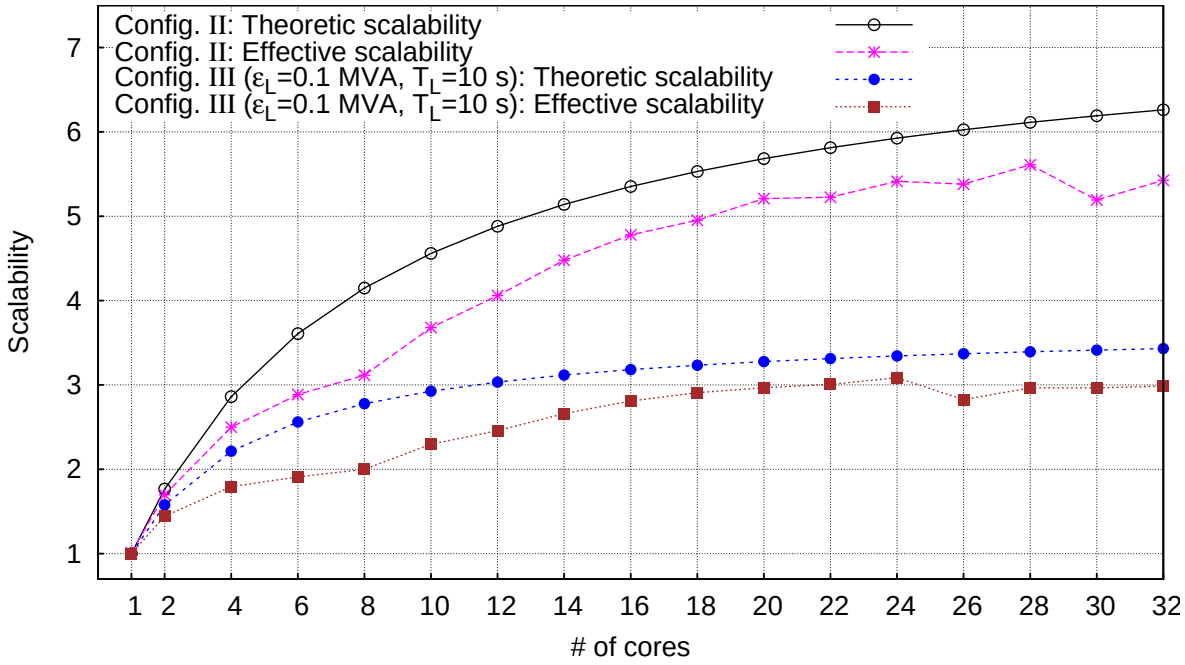


Figure 4.34: HQ: Effective VS Theoretic scalability

is at $t = 94$ s due to the shifted ASRT action. By simulating several scenarios with different latency tolerance values, it was found that an $\epsilon_L \leq 0.2$ MVA gives a good speedup while keeping the inaccuracy introduced at an acceptable level.

Table 4.5 shows the simulation time, speedup and maximum inaccuracy over all the bus voltages compared to the integrated method. From the sequential execution timings, it can be seen that all configurations offer some speedup. Configuration III offers the highest speedup in sequential execution at the cost of introducing the already mentioned error.

From the parallel execution timings of Table 4.5, it can be seen that all configurations offer significant speedup when parallelized: up to 8.8 times without any inaccuracy and up to 11.1 times when latency is used and $\epsilon_L \leq 0.2$ MVA. A more detailed view is offered in Fig. 4.33, where the speedup is shown as a function of the number of cores used.

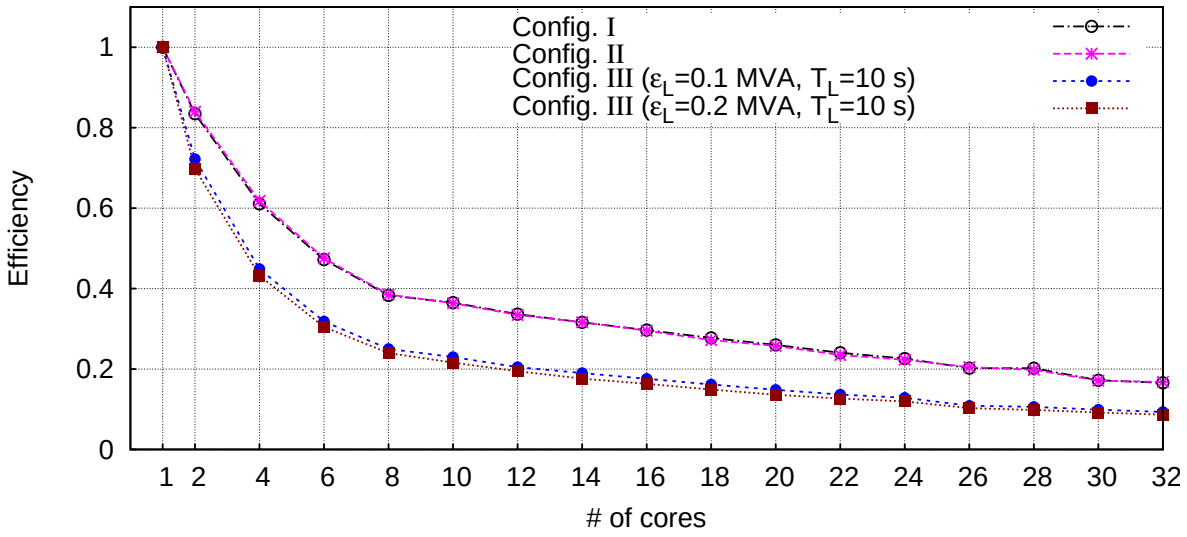


Figure 4.35: HQ: Efficiency computed with Eq. 2.3

Table 4.6 shows the time profiling of the sequential execution ($M = 1$), with the percentage of time spent in each block of the algorithm in Fig. 4.3. As expected, using Config. III leads to lower percentage of parallel work as the DAE models of injectors are replaced by simple sensitivity models in *BLOCKS A* and *C.* Figure 4.34 shows the scalability (computed with Eq. 2.1) against the theoretic scalability (computed with Eq. 2.5 and T_P and T_S of Table 4.6). The difference between them is due to the OHC of the implementation. In addition, Fig. 4.35 shows the parallelization efficiency using Eq. 2.3. It can be seen that Configs. I and II are parallelized more efficiently than Config. III. This is due to the higher percentage of parallel work in the former and the more difficult load balancing in the latter (as the amount of work performed by each task is unpredictable and leads to imbalances between the chunks used). The values given in Table 4.6 should be compared carefully as they relate to different execution times (as seen in Table 4.5).

Figure 4.36 shows the number of active injectors during the simulation with Config. III. It can be seen that in the short-term all injectors remain active, thus the main speedup in this period comes from the parallelization of the algorithm. In the long-term, and as the electromechanical oscillations fade, the injectors start switching to latent and decrease the computational burden of treating the injectors as well as the percentage of work in the parallel sections. Hence, in this part of the simulation, the main source of acceleration is latency. Consequently, the two sources of acceleration complement each other as they perform better at different parts of the simulation. This is the reason why, even though Config. III has less scalability, it is still the fastest in parallel execution (see Fig. 4.33).

Finally, Fig. 4.37 shows the real-time performance of the algorithm with Config. II. When the wall time curve is above the real-time line, then the simulation is lagging; otherwise, the simulation is faster than real-time and can be used for more demanding applications, like look-ahead simulations, training simulators or hardware/software in the loop. On this power

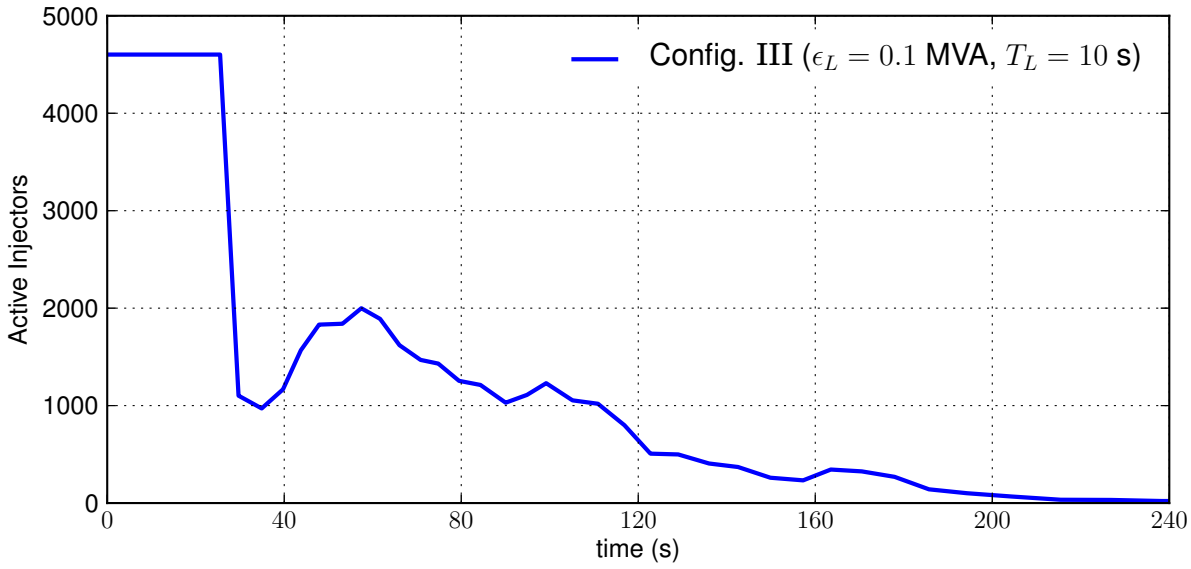


Figure 4.36: HQ: Number of active injectors

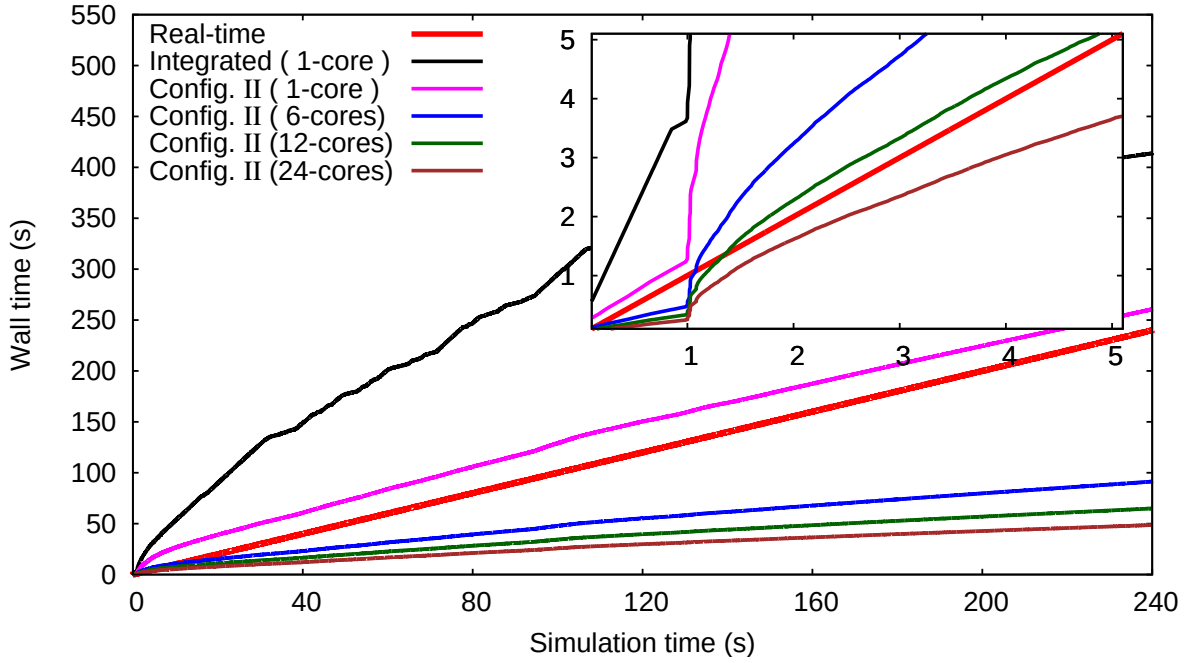


Figure 4.37: HQ: Real-time performance of algorithm

system, the algorithm performs faster than real-time when executed on 24 or more cores. The real-time performance of all systems will be further discussed in Section 4.9.4.3.

4.9.3 Pegase system

This is the largest test system considered in this work with 146239 DAEs. Decomposing the system leads to a network sub-domain, with 15226 buses, and $N = 10694$ injectors. The disturbance illustrated here consists of a bus bar fault, lasting five cycles, that is cleared

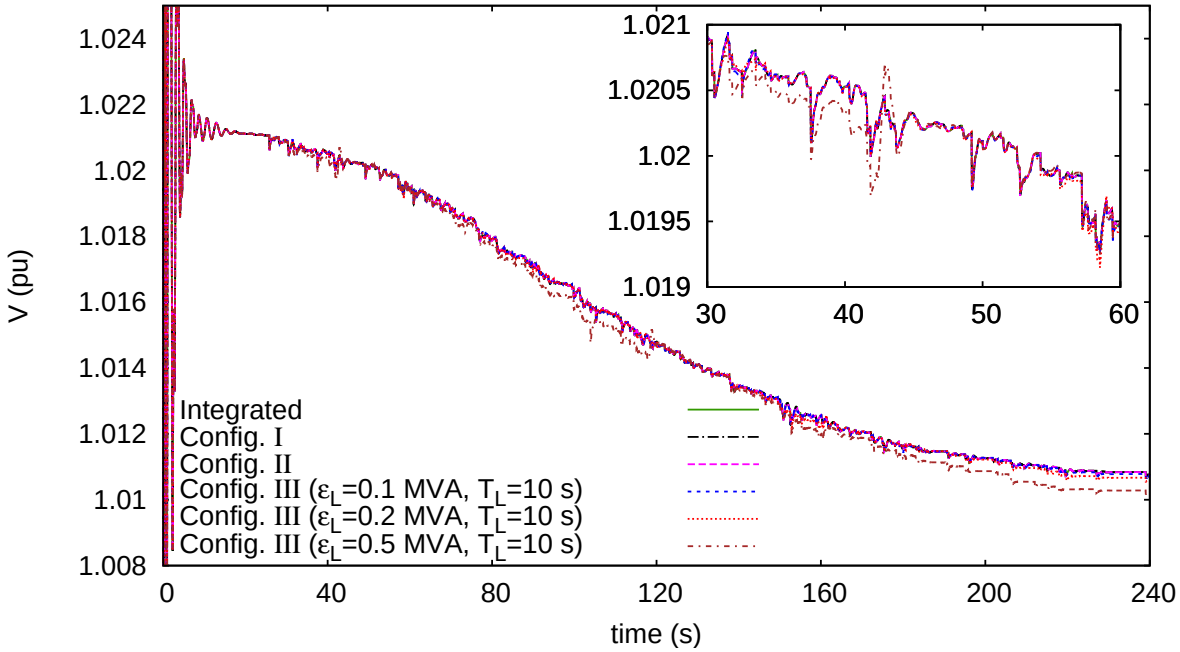


Figure 4.38: Pegase: Voltage evolution at bus $F0322411$

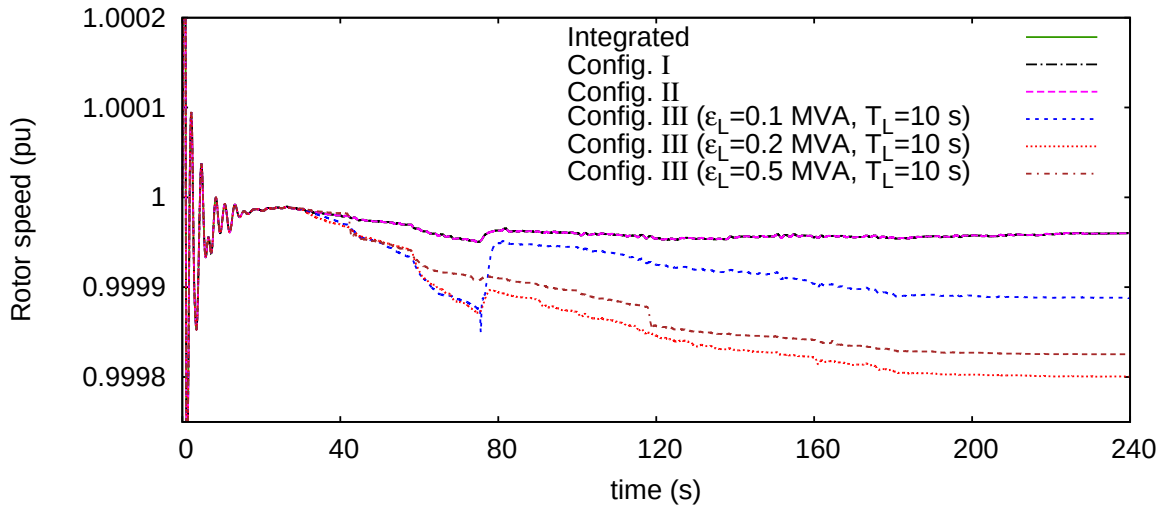


Figure 4.39: Pegase: Evolution of rotor speed of generator $FRHY1689$

by opening two double-circuit lines. The system is simulated over a period of 240 s with a time-step size of one cycle (20 ms).

Figures 4.38 and 4.39 show the voltage evolution of transmission bus $F0322411$ and the rotor speed of synchronous generator $FRHY1689$, respectively. This test case is stable in the long-term. After the electromechanical oscillations have died out, the system evolves under the effect of LTCs as well as OXLs. Thus, the decision about the stability of the system can only be made after the simulation of the whole time horizon. It should be noted that the discrepancy shown in Fig. 4.39 is insignificant (see scale) and is comparable to the

Table 4.7: Pegase: Execution times and inaccuracy in simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	1139.9 / -	- / -	-
Config. I	1228.6 / 0.9	221.6 / 5.1	0.0
Config. II	854.6 / 1.3	159.9 / 7.1	0.0
Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)	348.1 / 3.3	137.1 / 8.3	0.001
Config. III ($\epsilon_L = 0.2$ MVA, $T_L = 10$ s)	298.7 / 3.8	134.7 / 8.5	0.010
Config. III ($\epsilon_L = 0.5$ MVA, $T_L = 10$ s)	280.3 / 4.1	129.2 / 8.8	0.015

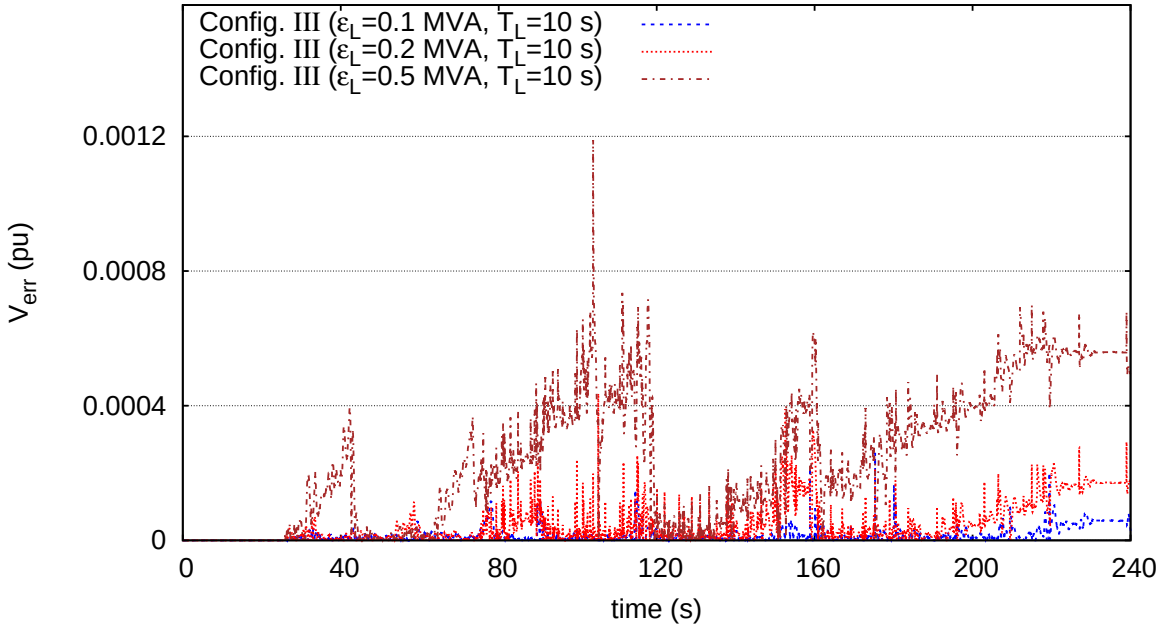


Figure 4.40: Pegase: Absolute voltage error on bus $F0322411$

convergence threshold (in this simulation 10^{-4}).

Table 4.7 shows the simulation time, speedup, and maximum inaccuracy over all bus voltages compared to the integrated method. From the sequential execution timings, it can be seen that in this system, Config. I is 10% slower than the integrated while the localization techniques offers some significant speedup with Configs. II and III.

Figure 4.40 shows the absolute error on the voltage at transmission bus $F0322411$. By simulating several scenarios with different latency tolerance values, it was found that an $\epsilon_L \leq 0.2$ MVA gives a good speedup while keeping the inaccuracy introduced at an acceptable level for most power system applications.

From the parallel execution timings of Table 4.7, it can be seen that all configurations offer

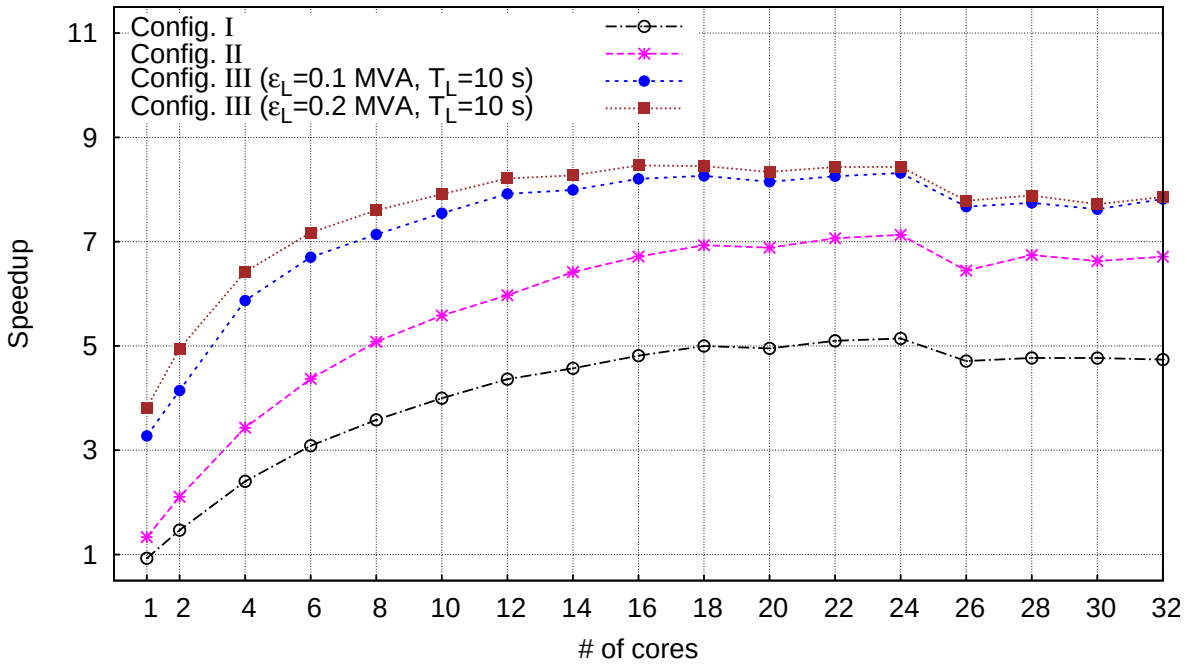


Figure 4.41: Pegase: Speedup computed with Eq. 2.2

Table 4.8: Pegase: Time profiling of sequential execution ($M = 1$)

	% of execution time		
	Config. I	Config. II	Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)
BLOCK A	12.12	8.63	6.8
BLOCK B	6.33	6.81	13.24
BLOCK C	66.76	66.85	46.47
BLOCK D	4.4	6.09	11.34
Remaining parallel	3.68	4.46	6.46
Remaining sequential	6.71	7.16	15.69
T_P (100-T_S)	86.96	86.03	71.07
T_S (BLOCK B+Rem. Seq.)	13.04	13.97	28.93

significant speedup when parallelized: up to 7.1 times without any inaccuracy and up to 8.8 times when latency is used. A more detailed view is offered in Fig. 4.41, where the speedup is shown as a function of the number of cores used for the simulation.

Table 4.8 shows the time profiling of the sequential execution, with the percentage of time spent in each block of the algorithm in Fig. 4.3. As noted previously, using Config. III leads to lower percentage of parallel work in *BLOCKS A* and *C*. Figure 4.42 shows the effective scalability (computed with Eq. 2.1) against the theoretic scalability (computed with Eq. 2.5 and the timings T_P and T_S of Table 4.8). In addition, Fig. 4.43 shows the parallelization efficiency using Eq. 2.3. It can be seen that Configs. I and II are parallelized more efficiently than Config. III due to the higher percentage of parallel work.

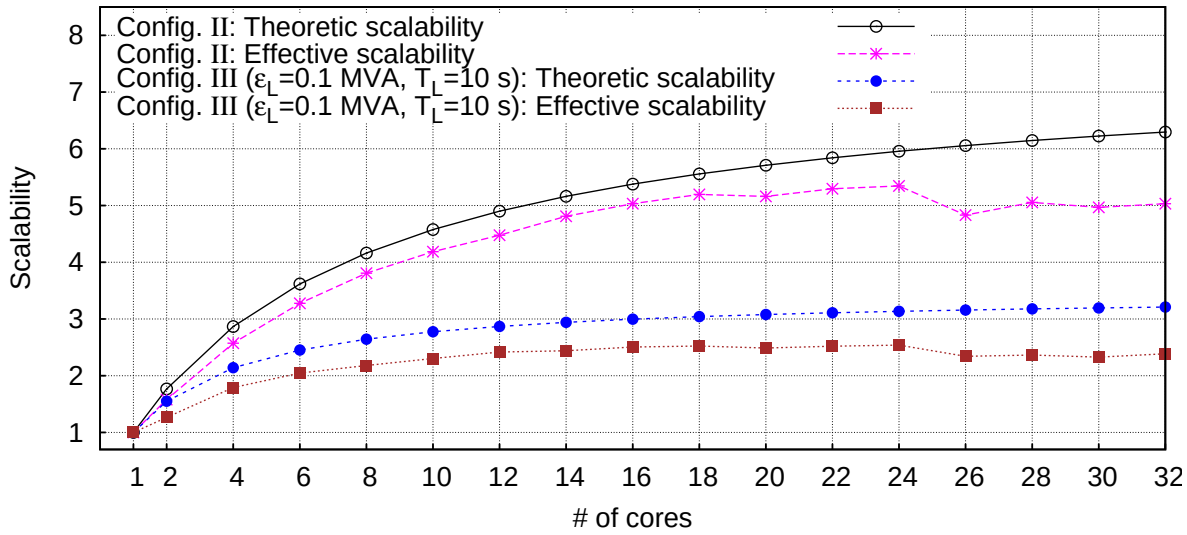


Figure 4.42: Pegase: Effective VS Theoretic scalability

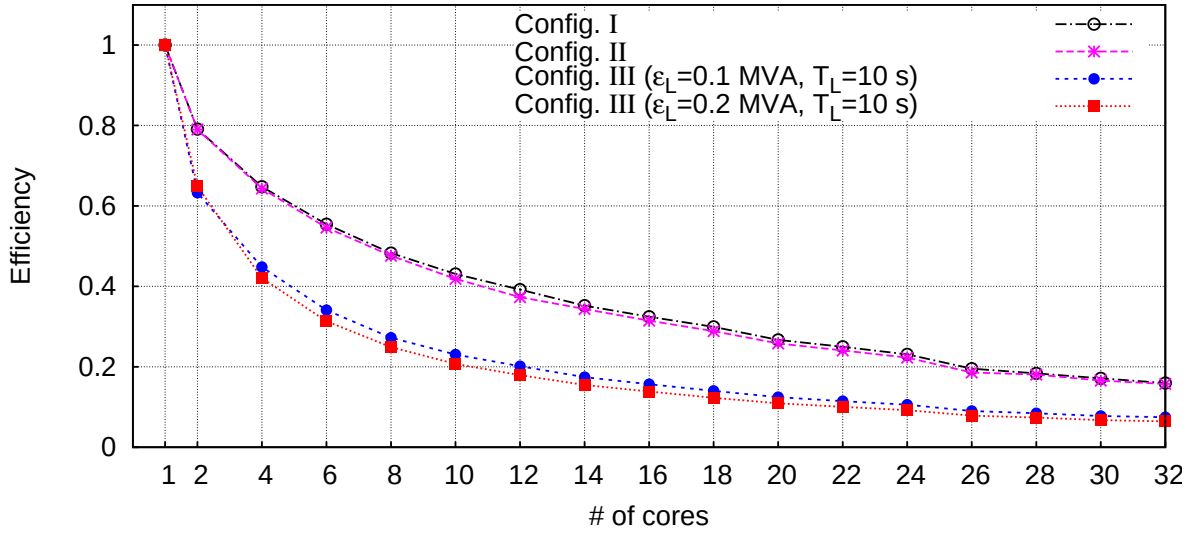


Figure 4.43: Pegase: Efficiency computed with Eq. 2.3

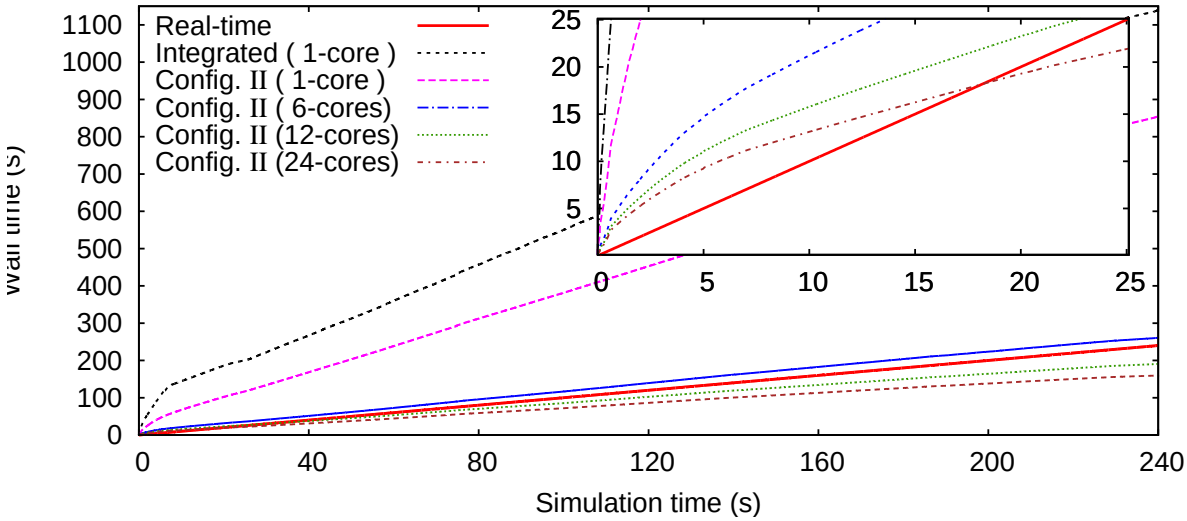


Figure 4.44: Pegase: Real-time performance of algorithm

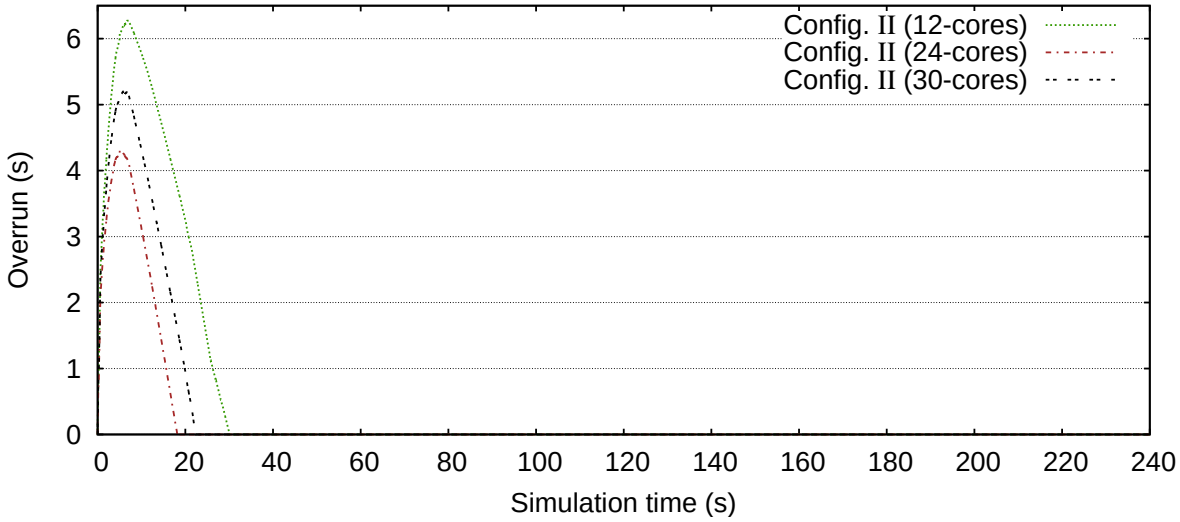


Figure 4.45: Pegase: Overrun of simulations

Finally, Fig. 4.44 shows the real-time performance of the algorithm with Config. II. Contrary to the previous test system, real-time performance is only achieved after approximately 17 s of simulation time. This means that some applications demanding "hard" real-time (at every time instant) are not possible for this 15000-bus system. However, applications with "soft" real-time demands (allow some limited overrun) are still possible. Figure 4.45 shows the overrun of the simulations, that is how much they are lagging from real-time. With $M = 24$, the overrun is limited to 4 s. The real-time performance of all systems will be further discussed in Section 4.9.4.3.

4.9.4 Discussion

This subsection presents a discussion concerning the sequential, parallel, and real-time performance of the proposed algorithm. Moreover, the test cases above are simulated with the UMA Machine 3 to show the performance on a standard office laptop.

4.9.4.1 Sequential performance

In sequential execution ($M = 1$), the main source of speedup for the proposed algorithm are the localization techniques presented in Section 4.6. Tables 4.3, 4.4, 4.5, and 4.7 show this performance in the sequential execution column.

The proposed DDM has some extra OHC compared to the integrated related to the bookkeeping and management of the sub-domains, the Schur-complement computation, etc. In addition, the memory management and the way of solving of the linear systems differ between the two algorithms. In the integrated, the entire Jacobian (see Eq. 4.20) is kept in memory and treated by the sparse solver. On the contrary, in the DDM, the injector matrices are kept separately in $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, and $\boldsymbol{C}_i$ and systems (4.6) are factorized and solved with the

LAPACK dense linear solvers (DGETRF, DGETRS). These small matrices are more likely to fit in the L1 cache memory at once, without the need of loading them in segments.

Configuration I does not use any localization techniques and follows the same Jacobian matrix updates as the integrated; it is in fact equivalent to solving (4.20). It can be seen that in the Nordic and HQ systems, the sequential execution of Config. I offers some speedup compared to the integrated. That means that the OHC of the proposed DDM is positively counteracted by the separate management and solution of the injectors. On the other hand, in the Pegase test case, Config. I is actually 10% slower than the integrated, meaning that the OHC of the DDM is higher.

On the contrary, Configs. II and III are always faster than the integrated as they exploit the localization techniques described in Section 4.6. These techniques strongly rely on the decomposed nature of the algorithm and would be very hard -if not impossible- to implement in the integrated method. First, all states are solved in the integrated system (4.20) no matter if some injectors have converged or not. Second, partially updating the factorized Jacobian of (4.20) is very complex and a specialized sparse linear solver would be necessary to achieve this. Finally, replacing the injector DAE models with linear equivalents would require to rebuild and factorize the integrated Jacobian matrix every time.

It has been shown that the proposed DDM, with the use of localization techniques, can provide significant acceleration even in sequential execution. This is important when executing on legacy single-core machines, or to increase the throughput of DSA schemes by simulating several contingencies concurrently using one core for each.

4.9.4.2 Parallel performance

One of the main advantages of DDMs is their parallelization potential. Tables 4.3, 4.4, 4.5, and 4.7 show the maximum speedup achieved by each configuration, while Figs. 4.21, 4.33, and 4.41 show the speedup as a function of the number of cores.

First, it can be seen that all the configurations gain from the parallelization. The highest speedup is achieved by the ones using the localization techniques. Thus, Configs. II and III already start from a large speedup in sequential execution (as seen in the previous subsection) and reach even higher speedups in parallel.

However, the *scalability* of the configurations is in reverse order. That is, the configurations that do not use localization techniques are parallelized more efficiently (see Figs. 4.35 and 4.43) and reach higher scalability values (see Figs. 4.34 and 4.42). The reason for this is explained in Section 4.8.1, and can be seen in practice in Tables 4.6 and 4.8. In these tables, the same test case was executed with Configs. I, II, and III and show that the *percentage* of time spent in the parallel segments (T_P) gets lower when acceleration techniques are used.

Furthermore, Figs. 4.34 and 4.42 show that the Pegase system, even though much larger in size, achieves the same scalability as HQ. That is because scalability is not proportional to

the size of the system but depends on T_P. Tables 4.6 and 4.8 show that this value is almost the same for both systems.

In practice, the proposed DDM exploits parallelization for the treatment of the injector subdomains. Thus, the higher the *percentage* of operations involving the injectors, the better the scalability. This has two contributing factors:

1. The number of network states (which is equal to the size of the sequentially treated reduced system of Eq. 4.12) compared to the number of injector states (which are treated in parallel). The biggest the ratio $\frac{\text{injector states}}{\text{network states}}$, the better the expected scalability. HQ has a ratio of 6.9 and Pegase of 4.8. Thus, one would expect HQ to have better scalability than Pegase.

2. However, as discussed thoroughly in Section 4.8.1, the computational cost of treating each injector is not equal, it depends on the type of injector, its nonlinearities, size, etc. Moreover, when localization techniques are used, the cost of each task becomes unpredictable and strongly depends on the contingency that is simulated.

Hence, while the ratio $\frac{\text{injector states}}{\text{network states}}$ provides an insight to the expected performance of the algorithm, scalability will eventually depend on the value of T_P which is not known beforehand and depends on the types of injectors, the use of localization techniques, the contingency simulated, the load-balancing efficiency, etc.

Finally, Figs. 4.34 and 4.42 show the theoretic and the effective scalability computed with Eqs. 2.5 and 2.1, respectively. It can be seen that the effective scalability is always smaller than the theoretic one and the difference between them (given by Eq. 4.30) increases for higher number of cores. This phenomenon is explained in Sections 2.5.2 and 4.8.3. Although it is possible in parallel implementations to achieve higher effective scalability that the theoretic due to differences in memory management (for example, the program has access to bigger total cache memory when using more cores) [Gov10], this was not observed in any of the simulations performed with RAMSES.

4.9.4.3 Real-time performance

Fast dynamic simulations of large-scale systems can be used for operator training and testing global control schemes implemented in Supervisory Control and Data Acquisition (SCADA) systems. In brief, measurements (such as the open/closed status from a switch, power flow, voltage, current, etc.) are transferred from Remote Terminal Units (RTUs) to the SCADA center through a communication system. These information are then visualized to the operators which take decisions for corrective actions to be communicated back to the RTUs. In rare applications, some remedial actions are computed automatically by closed-loop procedures. In modern SCADA systems the refresh rate (T_R) of these measurements is $2-5$ seconds [GSAR09].

Table 4.9: HQ: Execution times of UMA machines

	Sequential execution time ($M = 1$) (seconds/speedup)		Fastest parallel execution time (seconds/speedup)	
Machine (see Section 2.7)	2	3	2	3
Integrated (T_1^*)	244 / -	200 / -	- / -	- / -
Config. I	170 / 1.4	150 / 1.3	143 / 1.7	79 / 2.5
Config. II	139 / 1.8	121 / 1.7	107 / 2.3	67 / 3.0
Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)	125 / 2.0	58 / 3.5	119 / 2.1	39 / 5.1
Config. III ($\epsilon_L = 0.2$ MVA, $T_L = 10$ s)	108 / 2.3	44 / 4.6	94 / 2.6	31 / 6.5
Config. III ($\epsilon_L = 0.5$ MVA, $T_L = 10$ s)	88 / 2.8	38 / 5.3	86 / 2.8	27 / 7.4

The simulator in these situations takes on the role of the real power system along with the RTU measurements and the communication system. It needs to provide the simulated "measurements" to the SCADA system with the same refresh rate as the real system. Thus, the concept of "real-time" for these applications translates to time deadlines and some overruns are acceptable.

For the Nordic test system, the execution is always faster than real-time no matter the number of threads or the use of localization techniques. This is the case also for the HQ system when 24 cores are used in Config. II (see Fig. 4.37). Thus, for these systems, all possible refresh times (T_R) can be met. The Pegase test system however, exhibits some overruns. These are shown in Fig. 4.45, where the maximum overrun is 4 s with 24 cores in Config. II. This means that the simulator can still be used for time deadlines with $T_R \geq 4$ s.

Overall, a model of 8000 buses and 11000 injectors (totaling 75000 DAEs) was found to be the limit of the proposed implementation, on Machine 1, without overruns. This limit was found automatically by taking the Nordic system and gradually replacing its distribution loads with a series of DN systems shown Fig. B.1. Then, three different contingencies in the TN were simulated on the modified system while checking for overruns. The procedure stopped when overruns were detected even when using all the available cores of Machine 1.

4.9.4.4 Performance with UMA standard office laptops

For the previous simulations, Machine 1 was used to allow "scanning" through a varying number of cores and show the performance of the algorithm as a function of this. However, this algorithm can provide significant speedup even on smaller UMA standard office machines. Thus, the previous test cases were executed on Machines 2 and 3 to show the performance of the algorithm. The former has a dual-core while the latter a quad-core processor. The dynamic response and the inaccuracy introduced by latency (Config. III) are not presented as they are identical to the ones shown previously.

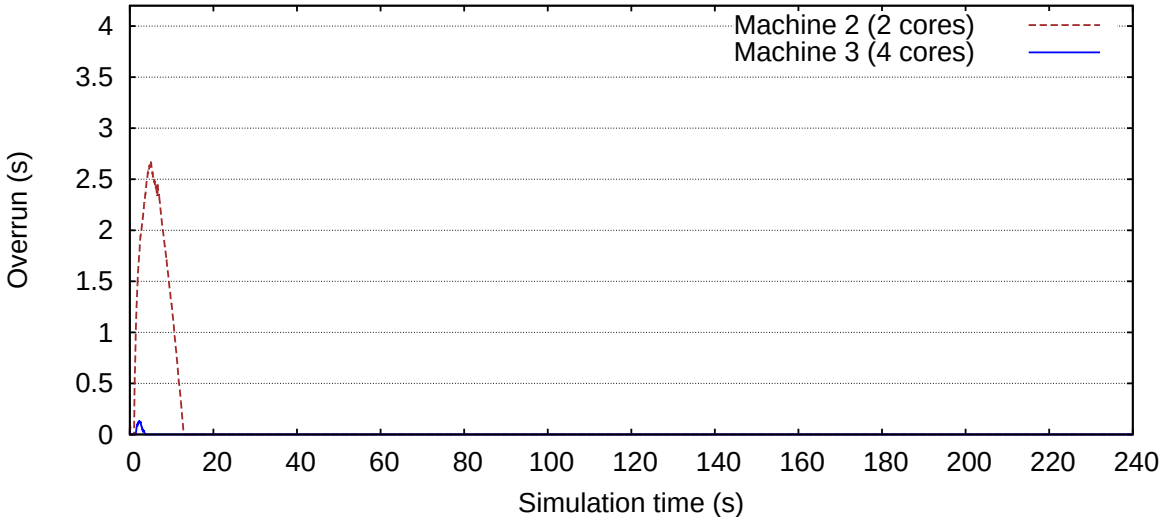


Figure 4.46: HQ: Overrun of the algorithm on UMA machines with Config. II

Table 4.10: Pegase: Execution times of UMA machines

	Sequential execution time ($M = 1$) (seconds/speedup)		Fastest parallel execution time (seconds/speedup)	
Machine (see Section 2.7)	2	3	2	3
Integrated (T_1^*)	585 / -	517 / -	- / -	- / -
Config. I	624 / 0.9	552 / 0.9	457 / 1.3	277 / 1.9
Config. II	435 / 1.3	383 / 1.4	340 / 1.7	212 / 2.4
Config. III ($\epsilon_L = 0.1$ MVA, $T_L = 10$ s)	173 / 3.4	148 / 3.5	150 / 3.9	105 / 4.9
Config. III ($\epsilon_L = 0.2$ MVA, $T_L = 10$ s)	145 / 4.0	125 / 4.1	137 / 4.3	94 / 5.5
Config. III ($\epsilon_L = 0.5$ MVA, $T_L = 10$ s)	141 / 4.2	119 / 4.4	129 / 4.5	89 / 5.8

Table 4.9 shows the execution times and speedup of simulating the test-case of Section 4.9.2, using the two laptop computers. Machine 2 (resp. 3) with Config. II, achieves a speedup of 1.8 (resp. 1.7) in sequential and 2.3 (resp. 3) in parallel execution, with a scalability of 1.3 (resp. 1.8).

Figure 4.46 shows the overrun of the simulation on both machines in parallel execution. It can be seen that on Machine 2 there is an overrun of 3.5 s, while in 3 the overrun is negligible, thus, real-time simulations are possible for this real 2500-bus (modeled with 35000 DAEs) system. Configuration III with $\epsilon_L = 0.2$ MVA offers a speedup of 2.6 (resp. 6.5) and simulates this long-term test case in 94 s (resp. 31 s).

Similarly, Table 4.10 shows the performance information of the UMA machines for the scenario of Section 4.9.3. With the full accurate Config. II, Machine 2 (resp. 3) achieves a speedup of 1.3 (resp. 1.4) in sequential and 1.7 (resp. 2.4) in parallel execution, with a

scalability of 1.3 (resp. 1.7). When the latency technique with $\epsilon_L = 0.2$ MVA is used, a speedup of 4.3 (resp. 5.5) is achieved and the test case is simulated in 137 s (resp. 94 s).

Overall, in this section it can be seen that the proposed algorithm provides significant speedup even on normal laptop computers. It allows to perform fast and accurate power system dynamic studies without the need of expensive equipment.

4.10 Summary

In this chapter, a parallel DDM-based algorithm has been proposed for the dynamic simulation of power systems. The algorithm partitions the network from the injectors attached to it, providing a star-shaped decomposition. The sub-domains formulated are treated independently and in parallel while their interface variables are updated using a Schur-complement approach. Furthermore, three localization techniques were presented to accelerate the simulation both in sequential and parallel execution.

First, the mathematical formulation of the proposed algorithm and localization techniques has been detailed. Next, a comparison of the proposed algorithm to the simultaneous approach was presented for the investigation of its convergence properties and how these are affected by the use of localization techniques. Then, the parallelization procedure was presented using the semantics of Chapter 2 and analyzing how the localization techniques and the OHC can affect the performance metrics. Subsequently, three test cases were presented involving a small, a medium and a large-scale test system, respectively. The accuracy and performance of the algorithm on all three cases was assessed compared to a simultaneous approach, implemented in the same software. Finally, a comparative overall assessment of the algorithms sequential, parallel, and real-time behavior was presented.

CHAPTER 5

Parallel two-level Schur-complement-based decomposition method

5.1 Introduction

The most noticeable developments foreseen in power systems involve Distribution Networks (DNs). Future DNs are expected to host a big percentage of renewable energy sources. The resulting challenge in power system dynamic simulations is to accurately model DNs and their participation to the bulk system dynamic behavior. This becomes compulsory as DNs are called upon to actively support the Transmission Network (TN) with an increasing number of Distributed Generators (DGs) and flexible loads participating in ancillary services through Smart Grid technologies.

In present-day dynamic security assessment of large-scale power systems, it is common to represent the bulk generation and higher voltage (transmission) levels accurately, while the lower voltage (distribution) levels are equivalenced. On the other hand, when studying the response of DNs, the TN is often represented by a Thévenin equivalent. The prime motivation behind this practice has been the lack of computational resources. Indeed, fully representing the entire power system network was historically impossible given the available computing equipment (memory capacity, processing speed, etc.) [KRF92]. Even with current computational resources, handling the entire, detailed model with hundreds of thousands of Differential and Algebraic Equations (DAEs) is extremely challenging [KRF92, GWA11].

As modern DNs are evolving with power electronic interfaces, DGs, active loads, and control schemes, more detailed and complex *dynamic* equivalent models would be needed to encompass the dynamics of DNs and their impact on the global system dynamics. A dynamic equivalent of a power system is a low-order dynamic model of the system, which is usually obtained by the reduction of a given full model [MMR10]. Some equivalencing approaches

reported in the literature are modal methods, synchrony (or coherency) methods [MMR10], and measurement or simulation-based methods [ANL+12]. Nevertheless, equivalent models inadvertently suffer from a number of drawbacks:

- The identity of the replaced system is lost. Faults that happen inside the DNs themselves cannot be simulated and individual voltages at internal buses, currents, controllers, etc. cannot be observed anymore. This makes it difficult to simulate controls or protections that rely on these values (e.g. fault ride through tripping of DGs).

- Most equivalent models target a specific type of dynamics (short or long-term, electromechanical oscillations, voltage recovery, etc.) and fail when used for other types. Hence, different types of simulations require different models. This adds an additional burden of maintaining and updating power system models when system take place.

- In most cases, the use or not of these equivalent models is decided *off-line*, when it is still unknown whether and how the contingency simulated will affect the DNs.

In this chapter, a parallel two-level Schur-complement-based DDM is proposed for the dynamic simulation of combined Transmission and Distribution (T&D) systems. First, the algorithm decomposes the combined system on the boundary between the TN and the DNs. This leads to the creation of several sub-domains, each defined by its own network and injectors. Then, a second decomposition scheme is applied within each sub-domain, splitting the network from the injectors, in a similar way to the single-level algorithm of Chapter 4. Finally, the solution of the sub-domain DAE systems is performed hierarchically with the interface variables being updated using a Schur-complement approach at each decomposition level. It will be shown that this algorithm can also be applied to sub-transmission networks, as long as some part of them is radial.

The proposed algorithm augments the performance of the simulation in two ways. First, the independent calculations of the sub-systems (on both decomposition levels) are parallelized providing computational acceleration. Second, the three localization techniques described in Section 4.6 are employed on both decomposition levels to avoid unnecessary computations and provide numerical acceleration.

The algorithm is first presented with a certain level of abstraction, focusing on its mathematical formulation. Next, the details concerning its implementation using the shared memory parallel computing model are presented. Finally, some results are shown using the expanded Nordic and Hydro-Québec systems presented in Section 1.3.

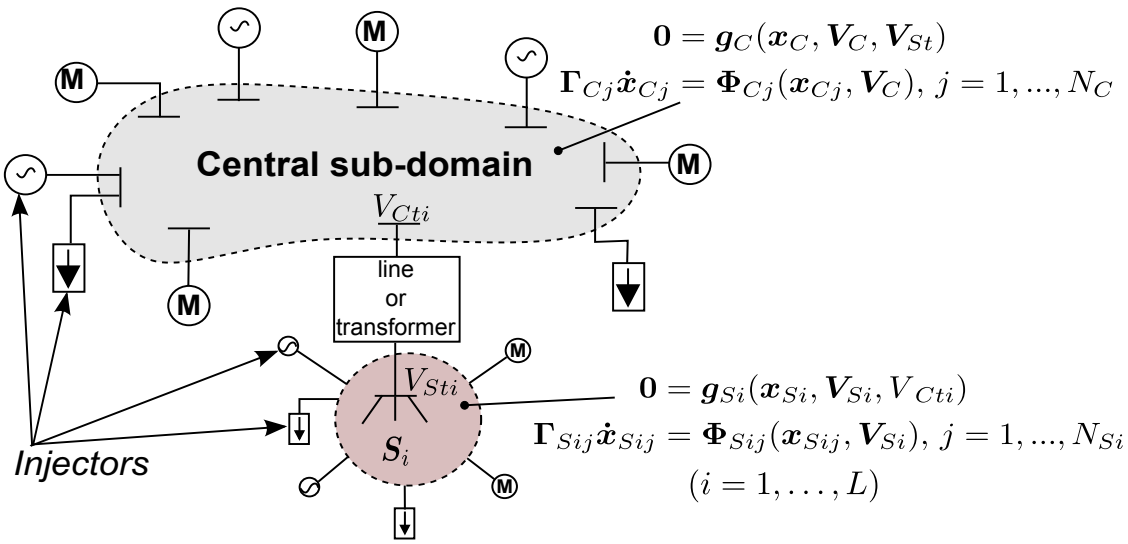


Figure 5.1: Two-level decomposed power system

5.2 Power system decomposition

5.2.1 First level of decomposition: Network

In the first level of decomposition, the power system is partitioned into sub-systems based on the topological representation of the network. Some techniques that have been proposed to partition power system networks have been summarized in Section 3.3.1. However, these techniques have the disadvantage that they can affect the convergence of the DDM. On the contrary, it was shown in Chapter 4 that a star-shaped non-overlapping partition scheme, combined with a Schur-complement treatment of the interface variables, can be reformulated as an equivalent simultaneous solution using an integrated Newton scheme. This has the benefit of retaining the good convergence properties of Newton methods and providing the theoretical basis to analyze the algorithm convergence.

For this reason, the network partitioning in the proposed algorithm aims for a star-shaped, topological-based, non-overlapping decomposition. That way, the power system network is decomposed into a *Central (C)* and several *Satellite sub-domains* (S_i) each connected to the Central *at one bus* as shown in Fig. 5.1.

Good candidate systems with this topology are combined transmission and distribution systems represented in detail, as DNs are usually attached to the TN at one bus through one or more parallel transformers. The star-shaped decomposition of these systems is trivial and can be based on voltage levels or over the distribution transformers.

However, some transmission systems can be also decomposed in this manner as long as some parts of the network are radial. In this case, the graph representation of the system can be used to extract the necessary topological information. An easy way to achieve this, is by recursively merging all buses (graph nodes) with only one network connection (graph edge), as presented in the following algorithm:

Algorithm 5.1 Computing star-shaped partition based on the network graph

```
nodeList=List of nodes with only one edge
while nodeList not empty do
    Merge all nodes in nodeList to their neighbor
    nodeList=List of nodes with only one edge
end while
```

Although this algorithm is not computationally efficient (it has a complexity of $O(R!)$, where R is the number of buses), it must only be executed once for each network. There is no need to revise the decomposition when topological changes are applied to the network, unless the changes destroy the star-shaped decomposition (for example a new line or transformer is put in service between two Satellite sub-domains).

Either from the electrical topology (voltage levels and distribution transformers) or with the use of Algorithm 5.1, the power system sketched in Fig. 5.1 is partitioned into the Central and L Satellite sub-domains, along with their injectors. This decomposition is reflected on the system of DAEs (1.1) as follows.

First, the DAE system describing the Central sub-domain with its injectors becomes:

$$\begin{aligned} \mathbf{0} &= \boldsymbol{g}_C(\boldsymbol{x}_C, \boldsymbol{V}_C, V_{St1}, \ldots, V_{StL}) \\ \boldsymbol{\Gamma}_C \dot{\boldsymbol{x}}_C &= \boldsymbol{\Phi}_C(\boldsymbol{x}_C, \boldsymbol{V}_C) \end{aligned} \tag{5.1}$$

and then, for the i-th Satellite sub-domain ($i = 1, \ldots, L$):

$$\begin{aligned} \mathbf{0} &= \boldsymbol{g}_{Si}(\boldsymbol{x}_{Si}, \boldsymbol{V}_{Si}, V_{Cti}) \\ \boldsymbol{\Gamma}_{Si} \dot{\boldsymbol{x}}_{Si} &= \boldsymbol{\Phi}_{Si}(\boldsymbol{x}_{Si}, \boldsymbol{V}_{Si}) \end{aligned} \tag{5.2}$$

where $\boldsymbol{x}_C$, $\boldsymbol{x}_{Si}$, $\boldsymbol{V}_C$, $\boldsymbol{V}_{Si}$, $\boldsymbol{\Gamma}_C$, and $\boldsymbol{\Gamma}_{Si}$ are the projections of $\boldsymbol{x}$, $\boldsymbol{V}$ and $\boldsymbol{\Gamma}$, defined in (1.1), on the Central and Satellite sub-domains respectively. V_{Sti} and V_{Cti} are the two bus voltages of the network connection between the Central and the i-th Satellite sub-domain (see Fig. 5.1).

The DAE systems (5.1) and (5.2) are coupled through the common variables $\boldsymbol{V}_{St} = [V_{St1}, \ldots, V_{StL}]$ and $\boldsymbol{V}_{Ct} = [V_{Ct1}, \ldots, V_{CtL}]$, involved in the equations of the network elements (lines, transformers, etc.) connecting the sub-domains. These $L + 1$ systems combined are mathematically equivalent to the original system (1.1).

5.2.2 Second level of decomposition: Injectors

Next, a second level of decomposition is applied that partitions the injectors from the sub-domain network, similarly to Chapter 4. For example, the j-th injector connected to the Central sub-domain (see Fig. 5.1) is described by a DAE system:

$$\boldsymbol{\Gamma}_{Cj} \dot{\boldsymbol{x}}_{Cj} = \boldsymbol{\Phi}_{Cj}(\boldsymbol{x}_{Cj}, \boldsymbol{V}_C), \;\; j = 1, \ldots, N_C \tag{5.3}$$

where N_C is the number of injectors attached on the Central sub-domain network, $\boldsymbol{x}_{Cj}$ and $\boldsymbol{\Gamma}_{Cj}$ are the projections of $\boldsymbol{x}_C$ and $\boldsymbol{\Gamma}_C$ (defined in Eq. 5.1) on the j-th injector. Thus, $\boldsymbol{x}_C = [\boldsymbol{x}_{C1} \; \ldots \; \boldsymbol{x}_{CN_C}]^T$ and $\boldsymbol{\Gamma}_C = diag[\boldsymbol{\Gamma}_{C1}, \ldots, \boldsymbol{\Gamma}_{CN_C}]$.

Therefore, system (5.1) becomes:

$$\begin{aligned} \mathbf{0} &= \boldsymbol{g}_C(\boldsymbol{x}_C, \boldsymbol{V}_C, \boldsymbol{V}_{St}) \\ \boldsymbol{\Gamma}_{Cj}\dot{\boldsymbol{x}}_{Cj} &= \boldsymbol{\Phi}_{Cj}(\boldsymbol{x}_{Cj}, \boldsymbol{V}_C), \;\; j = 1, ..., N_C \end{aligned} \tag{5.4}$$

Similarly, system (5.2) becomes ($i = 1, \ldots, L$):

$$\begin{aligned} \mathbf{0} &= \boldsymbol{g}_{Si}(\boldsymbol{x}_{Si}, \boldsymbol{V}_{Si}, V_{Cti}) \\ \boldsymbol{\Gamma}_{Sij}\dot{\boldsymbol{x}}_{Sij} &= \boldsymbol{\Phi}_{Sij}(\boldsymbol{x}_{Sij}, \boldsymbol{V}_{Si}), \;\; j = 1, ..., N_{Si} \end{aligned} \tag{5.5}$$

where N_{Si} is the number of injectors attached to the S_i sub-domain network.

The rectangular components of the injector current $[i_y,\ i_x]$ are included in the differential algebraic state vector $\boldsymbol{x}$ and can be rewritten for the Central sub-domain as:

$$\boldsymbol{I}_C = \sum_{j=1}^{N_C} \boldsymbol{C}_{Cj}\boldsymbol{x}_{Cj} \tag{5.6}$$

where $\boldsymbol{C}_{Cj}$ is a trivial matrix with zeros and ones whose purpose is to extract the injector current components from $\boldsymbol{x}_{Cj}$. Similarly, for the S_i sub-domain:

$$\boldsymbol{I}_{Si} = \sum_{j=1}^{N_{Si}} \boldsymbol{C}_{Sij}\boldsymbol{x}_{Sij} \tag{5.7}$$

As in the decomposition of Chapter 4, the network and injector systems are coupled through the currents injected into the network and the voltages of the buses where the injectors are attached to.

5.3 Sub-system solution

For the solution of each sub-system, the techniques detailed in Section 1.2 are used. First, the injector DAE systems are algebraized using a differentiation formula (in this work the second-order BDF), to get the corresponding nonlinear algebraized systems. For example, (5.3) becomes:

$$\mathbf{0} = \boldsymbol{f}_{Cj}(\boldsymbol{x}_{Cj}, \boldsymbol{V}_C), \;\; j = 1, \ldots, N_C \tag{5.8}$$

At each discrete time instant, the nonlinear, discretized injector equations are solved simultaneously with the linear network equations using a Newton method. Thus, at the k-th Newton iteration, the following systems are solved:

$$\boldsymbol{D}_C^k\,\Delta\boldsymbol{V}_C^k - \underbrace{\sum_{j=1}^{N_C} \boldsymbol{C}_{Cj}\,\Delta\boldsymbol{x}_{Cj}^k}_{\Delta\boldsymbol{I}_C^k} - \sum_{i=1}^{L} \boldsymbol{E}_{Si}^k\,\Delta\boldsymbol{V}_{Si}^k = -\underbrace{\boldsymbol{g}_C(\boldsymbol{x}_C^{k-1}, \boldsymbol{V}_C^{k-1}, \boldsymbol{V}_{St}^{k-1})}_{\boldsymbol{g}_C^k} \tag{5.9}$$

$$\boldsymbol{A}_{Cj}^k\,\Delta\boldsymbol{x}_{Cj}^k + \boldsymbol{B}_{Cj}^k\,\Delta\boldsymbol{V}_C^k = -\underbrace{\boldsymbol{f}_{Cj}(\boldsymbol{x}_{Cj}^{k-1}, \boldsymbol{V}_C^{k-1})}_{\boldsymbol{f}_{C_j}^k}, \;\; j = 1, ..., N_C \tag{5.10}$$

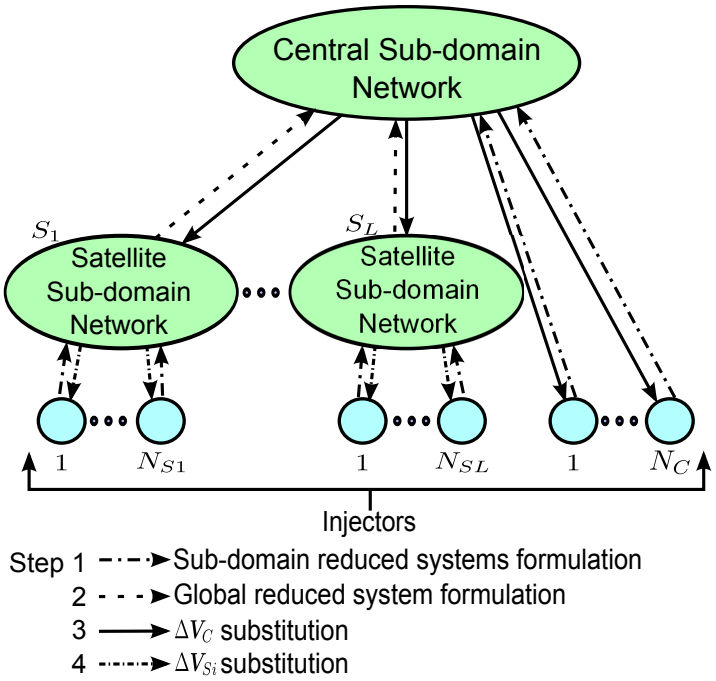


Figure 5.2: Hierarchical solution of two-level decomposed algorithm (four steps)

and ($i = 1, \ldots, L$):

$$\boldsymbol{D}_{Si}^k \Delta \boldsymbol{V}_{Si}^k - \underbrace{\sum_{j=1}^{N_{Si}} \boldsymbol{C}_{Sij} \Delta \boldsymbol{x}_{Sij}^k}_{\Delta \boldsymbol{I}_{Si}^k} + \boldsymbol{F}_{Si}^k \Delta \boldsymbol{V}_C^k = - \underbrace{\boldsymbol{g}_{Si}(\boldsymbol{x}_{Si}^{k-1}, \boldsymbol{V}_{Si}^{k-1}, V_{Cti}^{k-1})}_{\boldsymbol{g}_{Si}^k} \tag{5.11}$$

$$\boldsymbol{A}_{Sij}^k \Delta \boldsymbol{x}_{Sij}^k + \boldsymbol{B}_{Sij}^k \Delta \boldsymbol{V}_{Si}^k = - \underbrace{\boldsymbol{f}_{Sij}(\boldsymbol{x}_{Sij}^{k-1}, \boldsymbol{V}_{Si}^{k-1})}_{\boldsymbol{f}_{Sij}^k}, \quad j = 1, ..., N_{Si} \tag{5.12}$$

where $\boldsymbol{A}_{Cj}^k$ (resp. $\boldsymbol{A}_{Sij}^k$) is the Jacobian matrix of the j-th injector towards its own states and $\boldsymbol{B}_{Cj}^k$ (resp. $\boldsymbol{B}_{Sij}^k$) towards the voltages of its sub-domain. Finally, $\boldsymbol{E}_{Si}^k$ is the Jacobian of the Central sub-domain towards the voltages of the S_i sub-domain and $\boldsymbol{F}_{Si}^k$ is the Jacobian of the S_i sub-domain towards the voltage of the Central sub-domain.

The decomposed system results in $L + 1 + N_C + \sum_{i=1}^{L} N_{Si}$ linear systems (5.9)-(5.12) to be solved at each Newton iteration to compute the vectors $\boldsymbol{x}(t_n)$ and $\boldsymbol{V}(t_n)$. In the proposed algorithm, the solution is performed in an hierarchical manner as sketched in Fig. 5.2, using a Schur-complement approach at each iteration to treat the interface variables between sub-domains. This procedure is summarized below.

First, the sub-domain reduced systems are formulated by eliminating the injector states ($\boldsymbol{x}_{Cj}$ or $\boldsymbol{x}_{Sij}$) from the sub-domain network equations. This leads to reduced systems that involve *only* the sub-domain voltage states ($\boldsymbol{V}_C$ and $\boldsymbol{V}_{Si}$). This procedure is the same as described in Section 4.4 of the single-level algorithm. Twoports are treated as described in

the previous chapter with the limitation that they cannot be connected between two different Satellite sub-domains, as this destroys the star-shaped partition layout.

Second, the global reduced system is obtained by eliminating the Satellite sub-domain voltage states ($\boldsymbol{V}_{St}$) from the Central reduced system. This leads to a global reduced system that involves *only* the voltage states of the Central sub-domain ($\boldsymbol{V}_C$).

Then, the latter is solved and the computed Central sub-domain voltages ($\boldsymbol{V}_C$) are back-substituted into the sub-domain reduced systems. This decouples the solution of these systems which now involve only their own sub-domain voltage states ($\boldsymbol{V}_{Si}$). Thus, they can be solved independently.

Similarly, the sub-domain voltage states ($\boldsymbol{V}_C$ and $\boldsymbol{V}_{Si}$) are back-substituted into the injector equations, thus decoupling their solution as they now involve only their local states ($\boldsymbol{x}_{Cj}$ or $\boldsymbol{x}_{Sij}$). Hence, their solution can be also performed independently.

These steps are detailed in the following sub-sections.

5.3.1 Sub-domain reduced systems formulation

The sub-domain reduced systems are formulated by eliminating the injector states ($\boldsymbol{x}_{Cj}$ or $\boldsymbol{x}_{Sij}$) from the sub-domain network equation systems (5.9) and (5.11). This leads to reduced systems that involve *only* the sub-domain voltage states ($\boldsymbol{V}_C$ and $\boldsymbol{V}_{Si}$):

$$\left(\boldsymbol{D}_C^k + \sum_{j=1}^{N_C} \boldsymbol{C}_{Cj} \left(\boldsymbol{A}_{Cj}^k\right)^{-1} \boldsymbol{B}_{Cj}^k\right) \Delta\boldsymbol{V}_C^k - \sum_{i=1}^{L} \boldsymbol{E}_{Si}^k \Delta\boldsymbol{V}_{Si}^k = -\boldsymbol{g}_C^k - \sum_{j=1}^{N_C} \boldsymbol{C}_{Cj} \left(\boldsymbol{A}_{Cj}^k\right)^{-1} \boldsymbol{f}_{Cj}^k$$

$$\iff \quad \widetilde{\boldsymbol{D}}_C^k \, \Delta\boldsymbol{V}_C^k - \sum_{i=1}^{L} \boldsymbol{E}_{Si}^k \Delta\boldsymbol{V}_{Si}^k = -\widetilde{\boldsymbol{g}}_C^k \tag{5.13}$$

and ($i = 1, \ldots, L$):

$$\left(\boldsymbol{D}_{Si}^k + \sum_{j=1}^{N_{Si}} \boldsymbol{C}_{Sij} \left(\boldsymbol{A}_{Sij}^k\right)^{-1} \boldsymbol{B}_{Sij}^k\right) \Delta\boldsymbol{V}_{Si}^k + \boldsymbol{F}_{Si}^k \, \Delta\boldsymbol{V}_C^k = -\boldsymbol{g}_{Si}^k - \sum_{j=1}^{N_{Si}} \boldsymbol{C}_{Sij} \left(\boldsymbol{A}_{Sij}^k\right)^{-1} \boldsymbol{f}_{Sij}^k$$

$$\iff \quad \widetilde{\boldsymbol{D}}_{Si}^k \, \Delta\boldsymbol{V}_{Si}^k + \boldsymbol{F}_{Si}^k \, \Delta\boldsymbol{V}_C^k = -\widetilde{\boldsymbol{g}}_{Si}^k \tag{5.14}$$

As discussed in the previous chapter, the nonzero structures of the correction terms $\boldsymbol{C}_{Cj} \left(\boldsymbol{A}_{Cj}^k\right)^{-1} \boldsymbol{B}_{Cj}^k$ and $\boldsymbol{C}_{Sij} \left(\boldsymbol{A}_{Sij}^k\right)^{-1} \boldsymbol{B}_{Sij}^k$ depends on whether the component is an injector (attached to one bus) or a twoport (attached to two buses). Hence, the reduced system matrices $\widetilde{\boldsymbol{D}}_{Si}^k$ (resp. $\widetilde{\boldsymbol{D}}_C^k$) exhibit the sparsity pattern of $\boldsymbol{D}_{Si}^k$ (resp. $\boldsymbol{D}_C^k$) with some fill-in terms introduced by twoports.

5.3.2 Global reduced system formulation

Likewise, the global reduced system is formulated by eliminating the Satellite sub-domain voltage states ($\boldsymbol{V}_{Si}$) from the Central reduced system equations (5.13). This leads to a global

reduced system that involves *only* the voltage states of the Central sub-domain ($\boldsymbol{V}_C$):

$$\left(\widetilde{\boldsymbol{D}}_C^k + \sum_{i=1}^{L} \boldsymbol{E}_{Si}^k \left(\widetilde{\boldsymbol{D}}_{Si}^k\right)^{-1} \boldsymbol{F}_{Si}^k\right) \Delta \boldsymbol{V}_C^k = -\widetilde{\boldsymbol{g}}_C^k - \sum_{i=1}^{L} \boldsymbol{E}_{Si}^k \left(\widetilde{\boldsymbol{D}}_{Si}^k\right)^{-1} \widetilde{\boldsymbol{g}}_{Si}^k$$
$$\iff \quad \bar{\boldsymbol{D}}_C^k \, \Delta \boldsymbol{V}_C^k = -\bar{\boldsymbol{g}}_C^k \tag{5.15}$$

The global reduced system matrix $\bar{\boldsymbol{D}}_C^k$ maintains the sparsity pattern of $\widetilde{\boldsymbol{D}}_C^k$ as the elimination procedure is similar to that of an injector as described previously (attached to one bus). Moreover, the computations of $\boldsymbol{E}_{Si}^k \left(\widetilde{\boldsymbol{D}}_{Si}^k\right)^{-1} \boldsymbol{F}_{Si}^k$ and $\boldsymbol{E}_{Si}^k \left(\widetilde{\boldsymbol{D}}_{Si}^k\right)^{-1} \widetilde{\boldsymbol{g}}_{Si}^k$ are efficient as the matrices $\boldsymbol{E}_{Si}^k$ and $\boldsymbol{F}_{Si}^k$ are extremely sparse, given that each Satellite sub-domain is attached only to one bus in the Central sub-domain.

5.3.3 Back-substitution and solution

In this step, the global reduced system (5.15) is solved and the computed Central sub-domain voltage corrections ($\Delta \boldsymbol{V}_C^k$) are back-substituted into the sub-domain reduced systems (5.14). This decouples the solution of these systems which now involve only their sub-domain voltage states. Thus, they can be solved independently to acquire $\Delta \boldsymbol{V}_{Si}^k$.

Afterward, the computed sub-domain voltage corrections ($\Delta \boldsymbol{V}_{Si}^k$ and $\Delta \boldsymbol{V}_C^k$) are back-substituted in the injector equations (5.10) and (5.12), thus decoupling their solution as they now involve only their local state corrections ($\Delta \boldsymbol{x}_{Cj}^k$ or $\Delta \boldsymbol{x}_{Sij}^k$). Finally, the injector systems are solved independently.

After updating the state vectors, i.e. $\boldsymbol{V}_C^k = \boldsymbol{V}_C^{k-1} + \Delta \boldsymbol{V}_C^k$, $\boldsymbol{x}_C^k = \boldsymbol{x}_C^{k-1} + \Delta \boldsymbol{x}_C^k$, $\boldsymbol{V}_{Si}^k = \boldsymbol{V}_{Si}^{k-1} + \Delta \boldsymbol{V}_{Si}^k$ and $\boldsymbol{x}_{Sij}^k = \boldsymbol{x}_{Sij}^{k-1} + \Delta \boldsymbol{x}_{Sij}^k$, the convergence of all sub-systems is checked independently. If global convergence has been achieved, then the simulation proceeds to the next time instant, otherwise a new iteration $(k+1)$ is performed with the updated variables.

5.3.4 Base power selection

As mentioned in Section 1.2.5.2, in power system computations it is customary to scale the network variables and parameters by using a *per-unit system* [MBB08]. That is, a base power is selected (S_{base}) for the entire system and using the bus base voltage (usually set to its nominal value), all the network parameters (used to formulate the $\boldsymbol{D}$ matrix of Eq. 1.3) and variables ($\boldsymbol{V}$ and $\boldsymbol{I}$ of Eq. 1.3) are scaled accordingly. When simulating combined T&D systems, some very large injectors (such as large power plants attached to the TN) are solved together with smaller ones (such as small DGs attached to the DNs). Thus, using a single S_{base} for the entire system can lead to accuracy problems.

For example, let us assume a unique $S_{base} = 100$ MVA for the T&D system. A TN-connected generator producing 100 MVA at nominal voltage would have a current magnitude of 1 pu. On the other hand, a DN-connected DG producing 1 MVA at nominal voltage would have a current magnitude of 0.01 pu. If the convergence check of Eq. 1.16a is used with

$\epsilon_g = 0.001$, the latter generator will be solved at a smaller relative accuracy. To deal with this problem, a smaller ϵ_g has to be selected for *all* injectors to ensure the accurate solution of the smaller ones. In this way, however, the larger generator will be solved at a higher accuracy leading to larger computational burden due to more iterations.

Decomposing a T&D system as detailed in the previous sections of this chapter allows to select different S_{base} values in the various sub-domains. Thus, a smaller S_{base} can be used in the Satellite sub-domains (DNs) and a larger in the Central sub-domain (TN). In the same example as above, if $S_{baseC} = 100$ MVA is used for the TN and $S_{baseS} = 1$ MVA is for the DNs, the current output of both injectors will be at 1 pu, thus avoiding the previous problems.

5.4 Parallel algorithm

As discussed in Chapter 3, the main reason to employ a DDM is the parallelization potential of this type of algorithms. The overall proposed algorithm is sketched in Fig. 5.3 with the parallel segments being shaded. For each discrete time instant t_n, the parallel DDM described below is used to solve the sub-systems and obtain $\boldsymbol{V}(t_n)$ and $\boldsymbol{x}(t_n)$.

First, the injector and sub-network systems (5.9)-(5.12) are updated, their matrices are factorized, and their contributions to the reduced systems (5.13)-(5.14) are computed. The $L + 1 + N_C + \sum_{i=1}^{L} N_{Si}$ injector and network sub-systems are processed in parallel as there are no data dependencies between them during the update procedure. This can be seen in *BLOCK A* of Fig. 5.3.

Then, the $L + 1$ sub-domain reduced systems are formulated in parallel, as each one depends on its own injector models only. This is shown in *BLOCK B* of Fig. 5.3 where each parallel task updates one of the systems in (5.13)-(5.14).

Next, the global reduced system (5.15), whose contributing elements were computed in the previous BLOCK, is solved in *BLOCK C* to obtain the $\Delta\boldsymbol{V}_C$ corrections. As already noted in Chapter 4, this algorithm suffers from the sequential bottleneck of Schur-complement-based DDMs [Saa03]. However, due to the high sparsity of $\bar{\boldsymbol{D}}_C^k$ in the linear system (5.15) and its infrequent Jacobian update, this computational effort is bounded to 1-2% of the overall.

Afterward, the voltage corrections are introduced into the sub-domain reduced systems (5.14) and the data dependency between them is removed. Hence, these L sub-systems are solved in parallel, in *BLOCK D,* to compute the updated $\Delta\boldsymbol{V}_{Si}$ variables.

Then, the computed corrections $\Delta\boldsymbol{V}_C$ and $\Delta\boldsymbol{V}_{Si}$ are back substituted in Eqs. 5.10 and 5.12, thus decoupling the $N_C + \sum_{i=1}^{L} N_{Si}$ injector systems. The latter are solved in parallel as shown in *BLOCK E* to obtain the injector state corrections ($\Delta\boldsymbol{x}_{Cj}$ or $\Delta\boldsymbol{x}_{Sij}$). At the end of this block, the voltage ($\boldsymbol{V}$) and differential-algebraic state ($\boldsymbol{x}$) vectors are updated.

Finally, in *BLOCK F,* the convergence of the sub-systems is checked in parallel using the updated states. If all sub-systems have converged, the algorithm proceeds to the next time instant, otherwise, a new parallel solution ($k + 1$) is initiated.

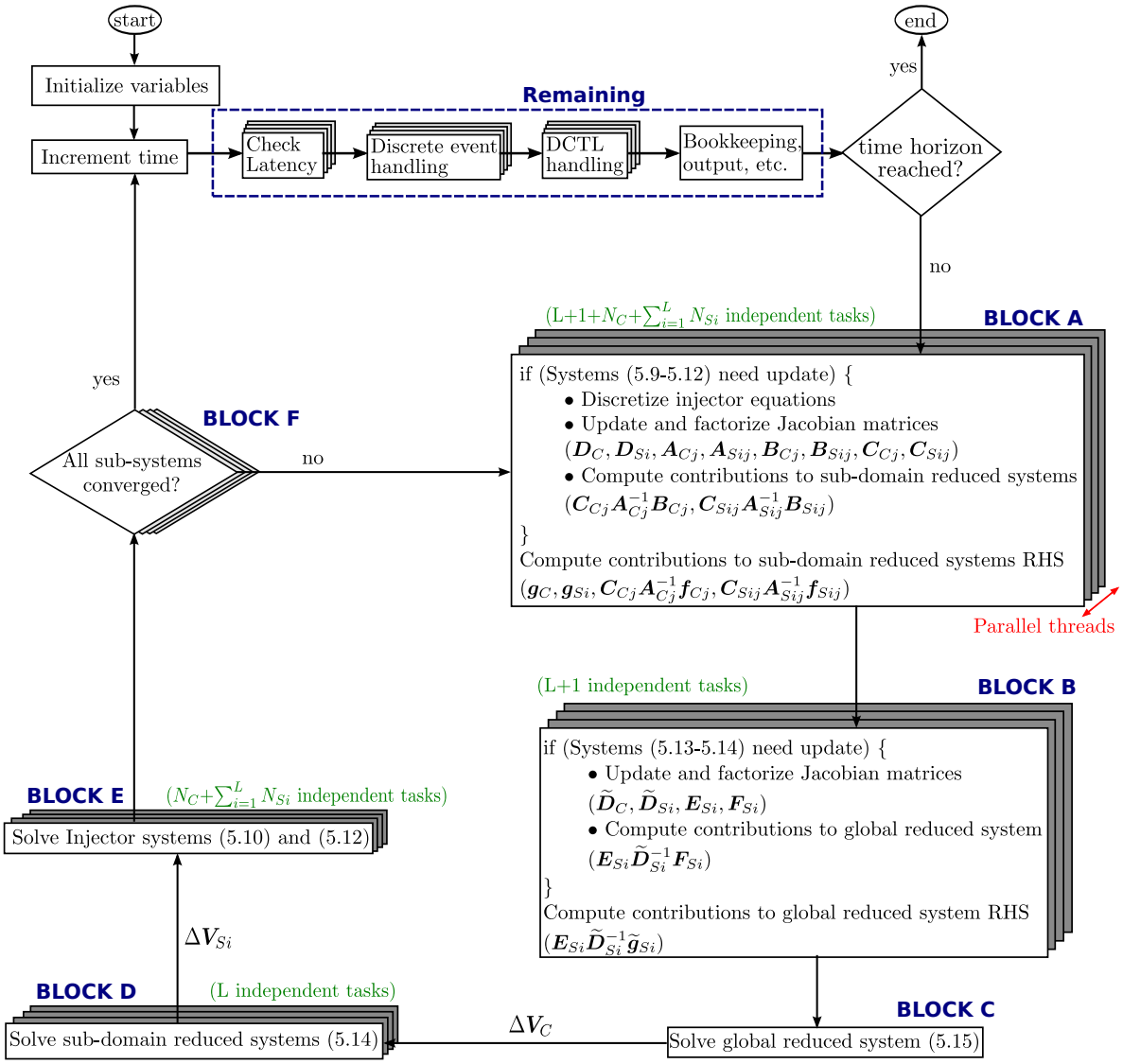


Figure 5.3: Two-level parallel solution algorithm

5.5 Localization techniques

The use of localization techniques was discussed in Section 4.6. Here, these concepts will be revisited and applied to the proposed two-level Schur-complement DDM. Figure 5.4 shows how the algorithm is modified with these techniques. In the following sub-sections these changes will be detailed.

5.5.1 Skipping converged sub-systems

The idea is similar to the one presented in Section 4.6.1, but applied at both decomposition levels. It is used within one discretized time instant solution to stop computations of injector or sub-domain reduced systems whose DAE models have already been solved with the desired

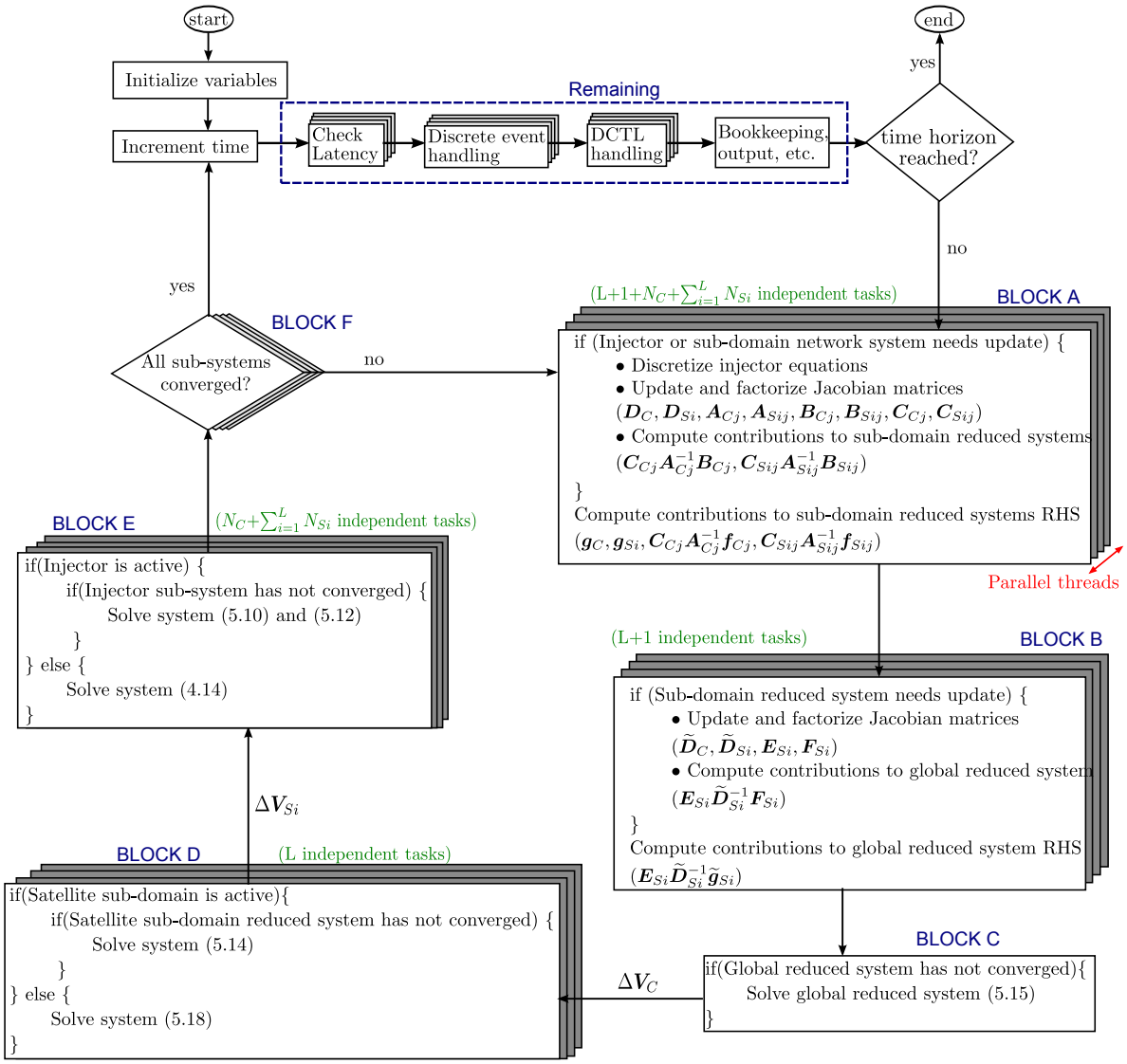


Figure 5.4: Two-level parallel solution algorithm with localization techniques

tolerance. That is, after one decomposed solution, the convergence of each injector and sub-domain reduced system is checked individually (*BLOCK F*). If the convergence criterion is satisfied, then the specific sub-system is flagged as converged and for the remaining iterations of the current time instant it is not solved. Nevertheless, its mismatch vector ($\boldsymbol{f}_{Cj}^k$, $\boldsymbol{f}_{Sij}^k$, $\widetilde{\boldsymbol{g}}_{Si}^k$, or $\bar{\boldsymbol{g}}^k{}_C$) is monitored to guarantee that it remains converged. This technique decreases the computational effort within one discretized time instant without affecting the accuracy of the solution.

5.5.2 Asynchronous update of injector or sub-domain reduced matrices

Next, taking advantage of the fact that each sub-domain is solved using the VDHN presented in Section 1.2.5.2, the sub-system update criteria are decoupled and their matrices (such as

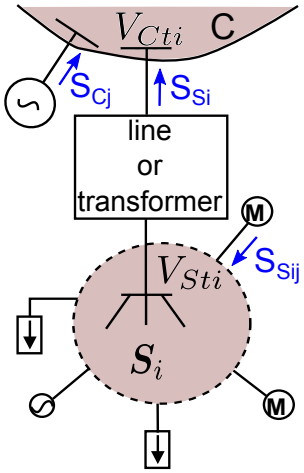


Figure 5.5: Latency applied at both decomposition levels (S denotes apparent power)

$\boldsymbol{D}_C$, $\boldsymbol{D}_{Si}$, $\boldsymbol{A}_{Cj}$, $\boldsymbol{A}_{Sij}$, $\boldsymbol{B}_{Cj}$, $\boldsymbol{B}_{Sij}$, etc.), as well as their Schur-complement contributions and the sub-domain reduced systems, are updated asynchronously. In this way, sub-domains at both decomposition levels, which converge fast keep the same matrices for many iterations and even time-steps, while sub-domains which converge slower update their matrices more frequently. Thus, *BLOCKS A* and *B* of Fig. 5.3 are replaced by those in Fig. 5.4.

The same update criteria as in Chapter 4 are used. That is, if a sub-system has not converged after five iterations of the algorithm presented in Fig. 5.3, its matrices and Schur-complement terms are updated. Moreover, an update of the matrices is triggered when a change in the equations of the sub-system is detected. Of course, after a severe event in the system (such as a short-circuit, the tripping of a branch or a generator, etc.) or when the time step size used for the discretization is changed, an update of all the matrices, the Schur-complement terms and the reduced systems is forced to avoid convergence problems.

5.5.3 Latency

The use of latency to decrease the computational burden has been investigated in Section 4.6.3. This technique can be extended and applied at both levels of the hierarchical scheme detailed in this chapter. Figure 5.5 shows the i-th Satellite sub-domain with its injectors. First, as in the single-level algorithm, the standard deviation of the apparent power of each injector (S_{Cj} or S_{Sij}) can be used to declare it latent and replace its dynamic model with the sensitivity-based one of Eq. 4.14. Second, the standard deviation of the apparent power S_{Si} exchanged between the Central and the i-th Satellite sub-domain can be used to declare the entire sub-domain (including its injectors) as latent.

The use of latency on the injectors has been thoroughly analyzed in the previous chapter. For the Satellite sub-domains, the same metrics (4.15)-(4.19) and switching procedure of Algorithm 4.1 can be used based on S_{Si}. The sensitivity model used for equivalencing the Satellite sub-domain is developed similarly to (4.14), but starting from the Satellite sub-domain re-

duced system (5.14). Hence, ignoring the internal dynamics, that is $\widetilde{\boldsymbol{g}}_{Si}(\boldsymbol{x}_{Si}^{k-1}, \boldsymbol{V}_{Si}^{k-1}, V_{Cti}^{k-1}) \simeq \mathbf{0}$, and solving for the sub-domain voltage variation $\Delta \boldsymbol{V}_{Si}$:

$$\Delta \boldsymbol{V}_{Si}^{k} \simeq -\left(\widetilde{\boldsymbol{D}}_{Si}^{k}\right)^{-1} \boldsymbol{F}_{Si}^{k} \Delta \boldsymbol{V}_{C}^{k} \tag{5.16}$$

Then, the corresponding terminal voltage variation $\Delta \boldsymbol{V}_{Sti}^{k}$ (see Fig. 5.5) is obtained by:

$$\Delta \boldsymbol{V}_{Sti}^{k} = -\boldsymbol{H}_{Si}\left(\widetilde{\boldsymbol{D}}_{Si}^{k}\right)^{-1} \boldsymbol{F}_{Si}^{k} \Delta \boldsymbol{V}_{C}^{k} = -\boldsymbol{G}_{Si} \Delta \boldsymbol{V}_{C}^{k} \tag{5.17}$$

where $\boldsymbol{H}_{Si}$ is a trivial matrix with zeros and ones whose purpose is to extract the terminal voltage variation from $\Delta \boldsymbol{V}_{Si}^{k}$, and $\boldsymbol{G}_{Si}$ is the sensitivity matrix relating the Satellite with the Central sub-domain voltage variation.

Selecting an arbitrary switching instant t^*, the linear relation (4.13) can be rewritten as:

$$\boldsymbol{V}_{Sti}(t_n) = \boldsymbol{V}_{Sti}(t^*) - \boldsymbol{G}_{Si}(t^*)\left[\boldsymbol{V}_{C}(t_n) - \boldsymbol{V}_{C}(t^*)\right] \tag{5.18}$$

for any discrete time $t_n \geq t^*$.

The linear equivalent model (5.18) involves only the voltage states of the Central and Satellite sub-domains as the interface variables between them are the voltages at the connection point. Moreover, it is important to note that the Schur-complement term contributed by the linear model (5.18) to the global reduced matrix $\bar{\boldsymbol{D}}_{C}$ of Eq. 5.15 is *the same* as the one of model (5.14). This means that switching from one model to the other doesn't require to recompute and factorize $\bar{\boldsymbol{D}}_{C}$.

Since latency is used at both decomposition levels, there needs to be a different priority between them. At the top level, if a Satellite sub-domain becomes latent then the injectors connected to it are not solved anymore. Thus, this level has priority over the lower level. On the lower level, injectors can individually become latent, as in the single-level algorithm, while other injectors in the same sub-domain remain active.

As discussed in Section 5.3.4, if the Satellite sub-domains represent DNs, the injectors connected to this level are usually much smaller in size than those connected to the TN. In such cases, it is useful to use a different latency tolerance for each injector depending on the sub-domain it is connected to. For example, a latency tolerance $\epsilon_L = 0.1$ MVA that was frequently used in Section 4.9 would lead most of the DN injectors to become latent immediately and remain so, as this value is comparable to their maximum apparent power. For this reason, a different latency tolerance is selected for each sub-domain. The selection of this value, however, should be in coordination with the selection of different sub-domain S_{base} values.

5.6 Effects of localization techniques on convergence

Similarly to the previous algorithm, the two-level Schur-complement-based DDM can be reformulated for the k-th iteration to the following equivalent integrated system:

$$\underbrace{\begin{pmatrix} \boldsymbol{A}_{S11} & \mathbf{0} & \cdots & \boldsymbol{B}_{S11} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{A}_{S12} & \cdots & \boldsymbol{B}_{S12} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ -\boldsymbol{C}_{S11} & -\boldsymbol{C}_{S12} & \cdots & \boldsymbol{D}_{S1} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{F}_{S1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{A}_{C1} & \mathbf{0} & \cdots & \boldsymbol{B}_{C1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{A}_{C2} & \cdots & \boldsymbol{B}_{C2} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{E}_{S1} & \cdots & -\boldsymbol{C}_{C1} & -\boldsymbol{C}_{C2} & \cdots & \boldsymbol{D}_{C} \end{pmatrix}^k}_{\boldsymbol{J}^k} \underbrace{\begin{pmatrix} \triangle \boldsymbol{x}_{S11} \\ \triangle \boldsymbol{x}_{S12} \\ \vdots \\ \triangle \boldsymbol{V}_{S1} \\ \vdots \\ \triangle \boldsymbol{x}_{C1} \\ \triangle \boldsymbol{x}_{C2} \\ \vdots \\ \triangle \boldsymbol{V}_{C} \end{pmatrix}^k}_{\triangle \boldsymbol{y}^k} = - \underbrace{\begin{pmatrix} \boldsymbol{f}_{S11} \\ \boldsymbol{f}_{S12} \\ \vdots \\ \boldsymbol{g}_{S1} \\ \vdots \\ \boldsymbol{f}_{C1} \\ \boldsymbol{f}_{C2} \\ \vdots \\ \boldsymbol{g}_{C} \end{pmatrix}^k}_{\boldsymbol{F}^k} \tag{5.19}$$

This system is equivalent to (1.15), presented in Section 1.2.5.2, when using the same discretization scheme and performing some row and column permutations. That is, the DDM-based solution presented above is mathematically equivalent to solving the set of nonlinear, discretized equations (5.9)-(5.12) with a quasi-Newton method. Thus, the observations of Section 4.7 about the convergence of the algorithm hold true in this case as well.

Similarly to the previous algorithm, the localization techniques modify the equivalent system. First, the matrices of each injector and of the network are not updated synchronously but according to the local convergence of each sub-system (see Section 5.5.2). Thus, the system is modified into:

$$\begin{pmatrix} \boldsymbol{A}_{S11}^{k_{s11}} & \mathbf{0} & \cdots & \boldsymbol{B}_{S11}^{k_{s11}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{A}_{S12}^{k_{s12}} & \cdots & \boldsymbol{B}_{S12}^{k_{s12}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ -\boldsymbol{C}_{S11}^{k_{s11}} & -\boldsymbol{C}_{S12}^{k_{s12}} & \cdots & \boldsymbol{D}_{S1}^{k_{s1}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{F}_{S1}^{k_{s1}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{A}_{C1}^{k_{C1}} & \mathbf{0} & \cdots & \boldsymbol{B}_{C1}^{k_{C1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{A}_{C2}^{k_{C2}} & \cdots & \boldsymbol{B}_{C2}^{k_{C2}} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{E}_{S1}^{k_C} & \cdots & -\boldsymbol{C}_{C1}^{k_{C1}} & -\boldsymbol{C}_{C2}^{k_{C2}} & \cdots & \boldsymbol{D}_{C}^{k_C} \end{pmatrix} \begin{pmatrix} \triangle \boldsymbol{x}_{S11} \\ \triangle \boldsymbol{x}_{S12} \\ \vdots \\ \triangle \boldsymbol{V}_{S1} \\ \vdots \\ \triangle \boldsymbol{x}_{C1} \\ \triangle \boldsymbol{x}_{C2} \\ \vdots \\ \triangle \boldsymbol{V}_{C} \end{pmatrix}^k = - \begin{pmatrix} \boldsymbol{f}_{S11} \\ \boldsymbol{f}_{S12} \\ \vdots \\ \boldsymbol{g}_{S1} \\ \vdots \\ \boldsymbol{f}_{C1} \\ \boldsymbol{f}_{C2} \\ \vdots \\ \boldsymbol{g}_{C} \end{pmatrix}^k \tag{5.20}$$

where $k_{Sij} \leq k$ ($i = 1, ..., L$, $j = 1, ..., N_{Si}$) and $k_{Cj} \leq k$ ($j = 1, ..., N_C$) is the iteration at which the j-th injector of the sub-domains was last updated. Similarly, $k_{Si} \leq k$ ($i = 1, ..., L$) and $k_C \leq k$ is the iteration at which the i-th Satellite or Central sub-domain network matrices were updated. These matrices are often kept constant over several iterations or even time steps. Therefore, this modified system can be treated as a quasi-Newton method with a special Jacobian update scheme. The error introduced to the Jacobian by the asynchronous

update of the sub-domain matrices is minimal, and while it can affect the convergence rate of the Jacobian, it does not affect the final solution.

Next, by also considering the skip-converged and latency techniques (see Sections 5.5.1 and 5.5.3) the system is modified to:

$$\begin{pmatrix} \boldsymbol{A}_{S11}^{k_{s11}} & \mathbf{0} & \cdots & \boldsymbol{B}_{S11}^{k_{s11}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{A}_{S12}^{k_{s12}} & \cdots & \boldsymbol{B}_{S12}^{k_{s12}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ -\boldsymbol{C}_{S11}^{k_{s11}} & -\boldsymbol{C}_{S12}^{k_{s12}} & \cdots & \boldsymbol{D}_{S1}^{k_{s1}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{F}_{S1}^{k_{s1}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{A}_{C1}^{k_{C1}} & \mathbf{0} & \cdots & \boldsymbol{B}_{C1}^{k_{C1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{A}_{C2}^{k_{C2}} & \cdots & \boldsymbol{B}_{C2}^{k_{C2}} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{E}_{S1}^{k_C} & \cdots & -\boldsymbol{C}_{C1}^{k_{C1}} & -\boldsymbol{C}_{C2}^{k_{C2}} & \cdots & \boldsymbol{D}_{C}^{k_C} \end{pmatrix} \begin{pmatrix} \triangle \boldsymbol{x}_{S11} \\ \triangle \boldsymbol{x}_{S12} \\ \vdots \\ \triangle \boldsymbol{V}_{S1} \\ \vdots \\ \triangle \boldsymbol{x}_{C1} \\ \triangle \boldsymbol{x}_{C2} \\ \vdots \\ \triangle \boldsymbol{V}_{C} \end{pmatrix}^k = - \begin{pmatrix} \boldsymbol{f}_{S11} \\ \boldsymbol{f}_{S12} \\ \vdots \\ \boldsymbol{g}_{S1} \\ \vdots \\ \boldsymbol{f}_{C1} \\ \boldsymbol{f}_{C2} \\ \vdots \\ \boldsymbol{g}_{C} \end{pmatrix}^k + \begin{pmatrix} \boldsymbol{r}_{S11} \\ \boldsymbol{r}_{S12} \\ \vdots \\ \boldsymbol{r}_{S1} \\ \vdots \\ \boldsymbol{r}_{C1} \\ \boldsymbol{r}_{C2} \\ \vdots \\ \boldsymbol{r}_{C} \end{pmatrix}^k \tag{5.21}$$

where:

$$\boldsymbol{r}_{Sij},\, \boldsymbol{r}_{Cj} = \begin{cases} \boldsymbol{f}_{Sij},\, \boldsymbol{f}_{Cj} & \text{if the j-th injector of a sub-domain is latent or converged} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\boldsymbol{B}_{Sij},\, \boldsymbol{B}_{Cj} = \begin{cases} \mathbf{0} & \text{if the j-th injector of a sub-domain is converged} \\ \boldsymbol{B}_{Sij},\, \boldsymbol{B}_{Cj} & \text{otherwise} \end{cases}$$

$$\boldsymbol{r}_{Si} = \begin{cases} \boldsymbol{g}_{Si} & \text{if the i-th Satellite sub-domain is latent or converged} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\boldsymbol{F}_{Si} = \begin{cases} \mathbf{0} & \text{if the i-th Satellite sub-domain is converged} \\ \boldsymbol{F}_{Si} & \text{otherwise} \end{cases}$$

$$\boldsymbol{C}_{Sij} = \begin{cases} \mathbf{0} & \text{if the i-th Satellite sub-domain is converged} \\ \boldsymbol{C}_{Sij} & \text{otherwise} \end{cases}$$

Hence, similar to the single-level algorithm of Chapter 4, an inexact Newton scheme is formulated and its convergence properties and assumptions can be examined as previously. The skip-converged technique is based on numerical criteria and uses the exact convergence tolerance to switch the sub-systems, thus it does not affect the final solution of the algorithm. If this technique is used on its own, the following condition:

$$\frac{\|\boldsymbol{r}^k\|}{\|\boldsymbol{F}(\boldsymbol{y}^k)\|} < \eta < 1 \qquad \forall k \geq 0 \tag{5.22}$$

can be easily checked to ensure the correctness of the decomposed algorithm (see Appendix A).

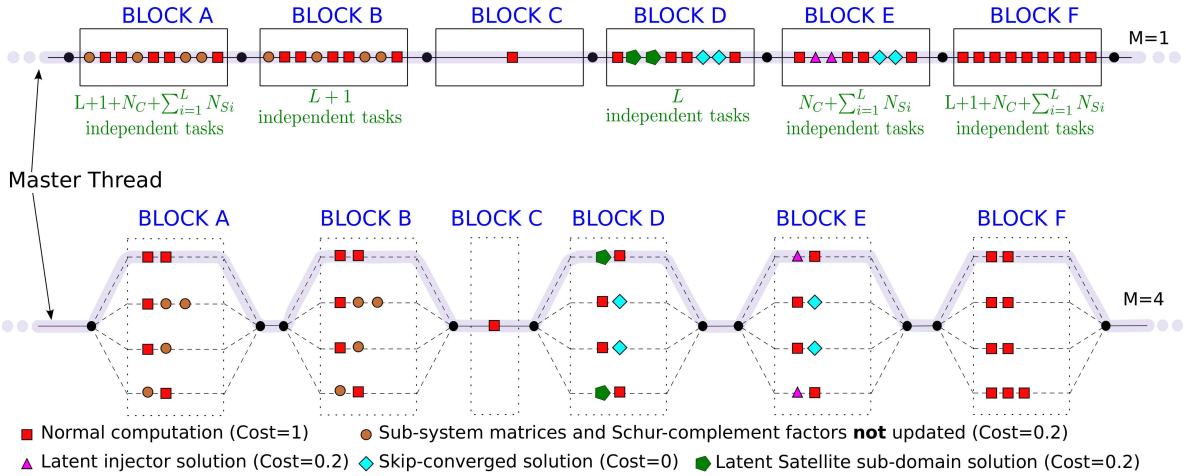


Figure 5.6: Fork-join pattern of algorithm in Fig. 5.3 with localization techniques

On the contrary, latency can introduce some inaccuracies in the simulation response which will be further analyzed from simulation results in Section 5.8.

5.7 Parallelization specifics

Figure 5.6 shows the fork-join parallel pattern of the DDM-based algorithm with the use of localization techniques. The observations detailed in Section 4.8, concerning the effect of localization techniques, load balancing, overhead costs, and profiling, hold true for this algorithm as well.

In this algorithm, there are two additional parallel *BLOCKS* compared to the single-level algorithm. *BLOCK B* takes care of the update and Satellite sub-domain reduced systems and *BLOCK D* of their solution. The sequential *BLOCK C* is much smaller than the corresponding system of the algorithm in Chapter 4, the reason being that the sparse linear system solved in the previous algorithm related to the size of the entire network, while the system in BLOCK C relates to the size of the Central sub-domain network only.

5.8 Experimental results

In this section a sample of simulation results are given, using the test systems summarized in Section 1.3. First, the contingency considered in each test system will be described. Next, several executions of the same simulation will be performed using a combination of simulation parameters. These are shown in Table 5.1.

As in the previous chapter, the non-decomposed approach with integrated VDHN scheme, described in Section 1.2.5.2, is used as a benchmark.

Both the proposed algorithm and the integrated are implemented in the academic simulation software RAMSES. The same models, convergence tolerance, algebraization method

Table 5.1: Options considered in the simulations

Configuration	Skip converged injectors	Update sub-domain matrices asynchronously	Exploit latency
I	✗	✗	✗
II	✓	✓	✗
IIIa	✓	✓	$\epsilon_{LC} = 0.1$ MVA, $\epsilon_{LS} = 0.01$ MVA, $T_L = 10$ s
IIIb	✓	✓	$\epsilon_{LC} = 0.2$ MVA, $\epsilon_{LS} = 0.02$ MVA, $T_L = 10$ s
IIIc	✓	✓	$\epsilon_{LC} = 0.5$ MVA, $\epsilon_{LS} = 0.05$ MVA, $T_L = 10$ s

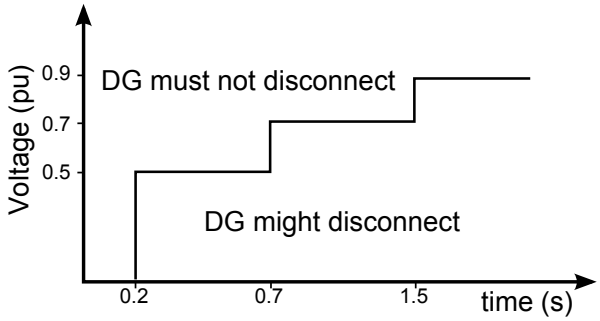


Figure 5.7: LVFRT capability curve of DGs [Sch08]

(second-order BDF), and way of handling the discrete events are used. For the solution of the sparse systems (the integrated Jacobian or the reduced systems of Eqs. 5.14 and 5.15), the sparse linear solver HSL MA41 [HSL14] was used. For the solution of the much smaller, dense injector linear systems (5.10) and (5.12), Intel MKL LAPACK library was used. The matrix update criteria are as follows: for the integrated and Config. I, all the matrices are updated every five iterations until convergence; for Configs. II and III, the matrices of *each* sub-domain are updated every five iterations unless convergence has already taken place. Finally, the convergence checks defined in Eqs. 1.16a and 1.17 are used, with $\epsilon_g = \epsilon_{frel} = \epsilon_{fabs} = 10^{-4}$. Keeping the aforementioned parameters and solvers of the simulation constant for both algorithms permits a (more) rigorous evaluation of the proposed algorithm performance.

The main investigations have been performed on Machine 1 (see Section 2.7), but a performance comparison with Machines 2 and 3 is given in Section 5.8.4.3. To better understand the behavior of the algorithm, some time profilings of the simulations are given in the following sections, while a numerical profiling is presented in Appendix D.

5.8.1 Nordic variant 1 system

This is a combined T&D system based on the Nordic system, which is expanded with 146 DNs. In total, the system includes 14653 buses, 15994 branches, 23 large synchronous ma-

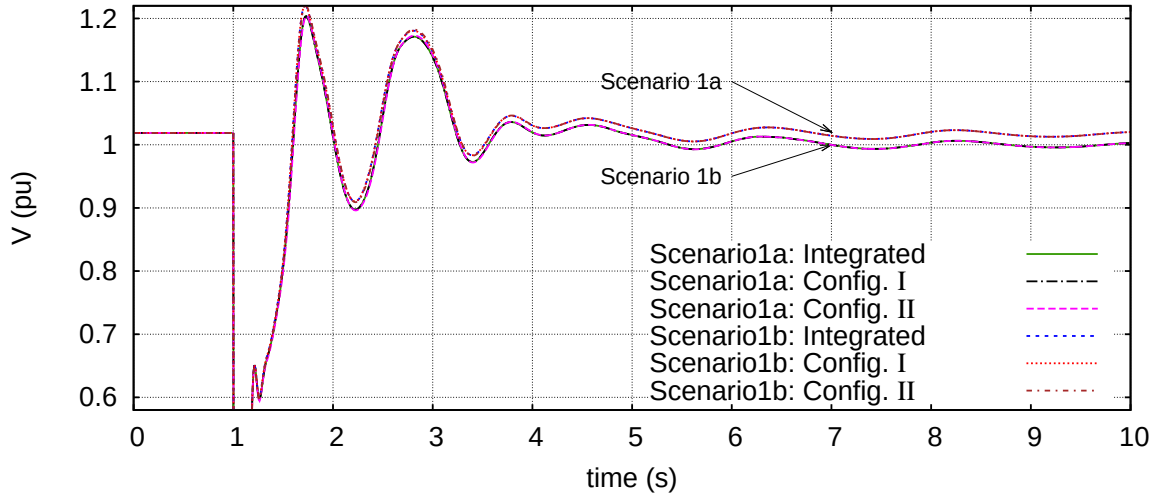


Figure 5.8: Nordic variant 1, Scenario 1: Voltage evolution at TN bus 1041

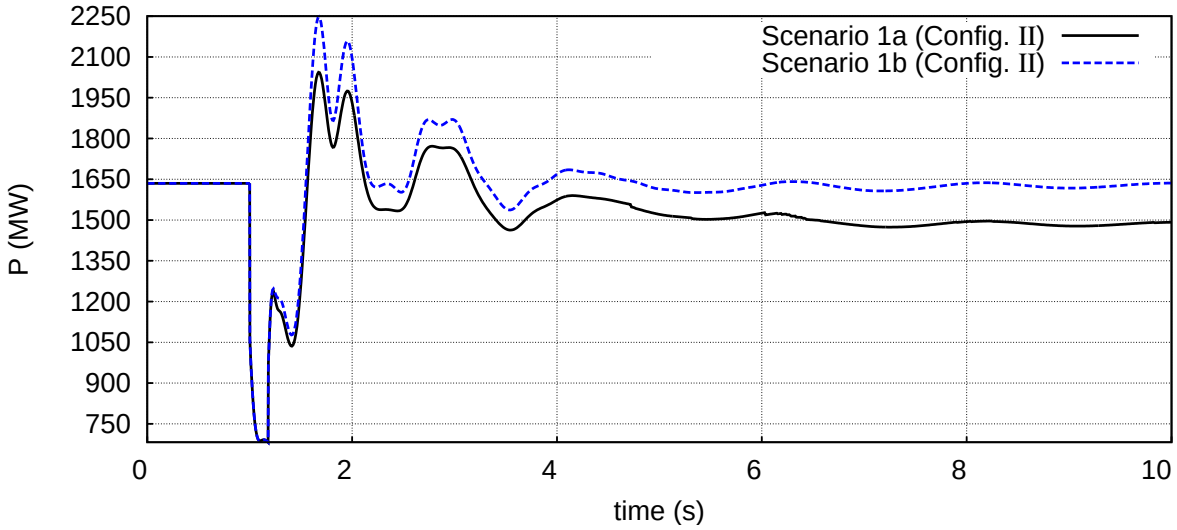


Figure 5.9: Nordic variant 1, Scenario 1: Total active power generated by DGs in all DNs

chines, 438 PVs, 730 WTs, and 19419 dynamically modeled loads. The resulting un-decomposed model has 143462 differential-algebraic states. A more detailed system description can be found in Section 1.3 and its one-line diagram in Fig. B.3.

The first decomposition is performed on the boundary between TN and DNs, thus creating the Central sub-domain with the TN and $L = 146$ Satellite sub-domains with the DNs. Next, each sub-domain is decomposed to its network and injectors. More specifically, $N_{S1} = N_{S2} = ... = 141$ and $N_C = 24$.

Two scenarios, respectively short and long-term, are considered for this system. Furthermore, each scenario is simulated twice. In the first simulation (referred to as Scenarios 1a and 2a), the DGs comply with the Low Voltage and Fault Ride Through (LVFRT) requirements sketched in Fig. 5.7, taken from [Sch08]. In the second simulation (referred to as Scenar-

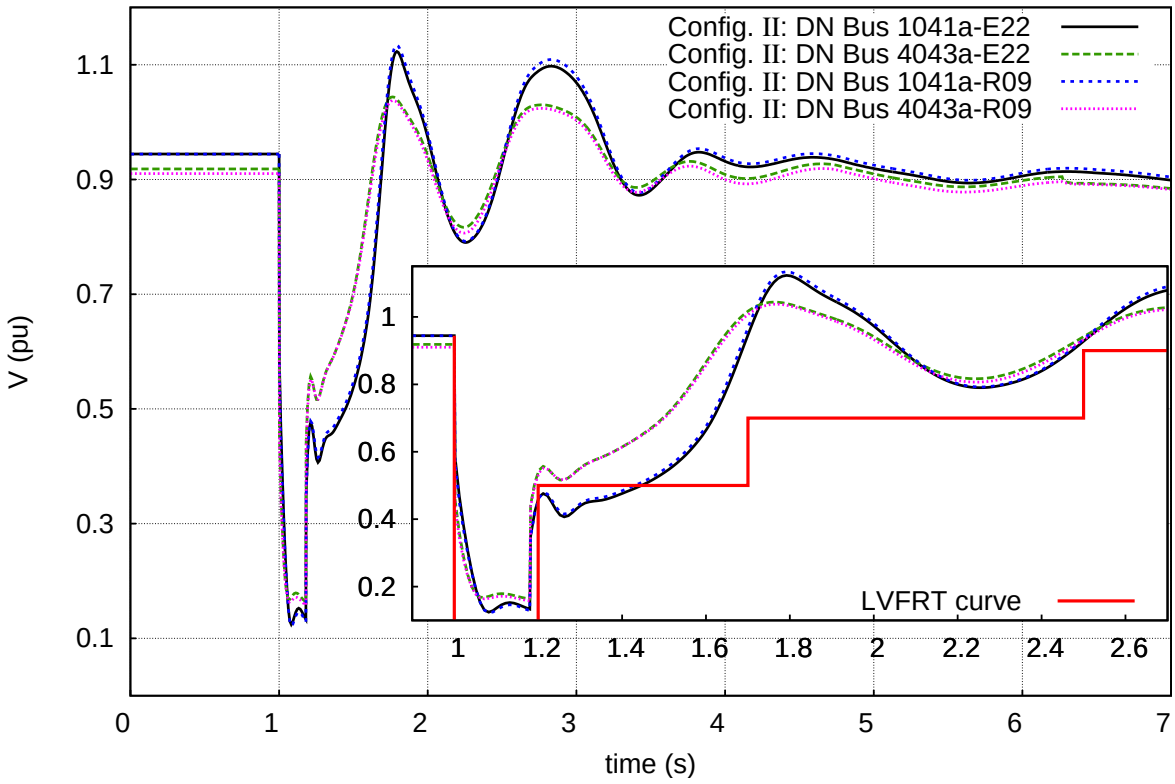


Figure 5.10: Nordic variant 1, Scenario 1a: Voltages at DN buses

ios 1b and 2b), the DGs remain connected to the system even if their terminal voltages fall below the limit shown in Fig. 5.7.

The nominal powers of the TN generators are hundreds of MVA while of the DN DGs are just a few MVA. Thus, as described in Section 5.3.4, different base powers are considered for the Central ($S_{baseC} = 100$ MVA) and Satellite ($S_{baseS} = 2$ MVA) sub-domains. For the integrated method, the smallest one is used for the entire system to ensure the accurate simulation of DN DGs. Moreover, different latency tolerance is used for injectors connected to the TN (ϵ_{LC}) and to a DN (ϵ_{LS}), as listed in Table 5.1.

5.8.1.1 Scenario 1: Short-term stability study

In this scenario, a short-term stability study is considered. The disturbance considered is an 8-cycle, three-phase, solid fault near the TN bus 4044, cleared by the opening the faulted line 4043-4044 in the CENTRAL area (see Fig. B.3). The system is simulated for 10 s with a time-step size of half a cycle (10 ms). Only the results acquired with Configs. I and II are shown in this scenario. The small simulation period, as well as the short-term nature of the dynamics considered make the use of latency unnecessary. Even if latency was considered, with $T_L \geq 10$ s, none of the components would become latent.

Figure 5.8 shows the voltage evolution at TN bus 1041 for Scenarios 1a and 1b. The integrated and DDM-based algorithms give exactly the same results and the system is short-

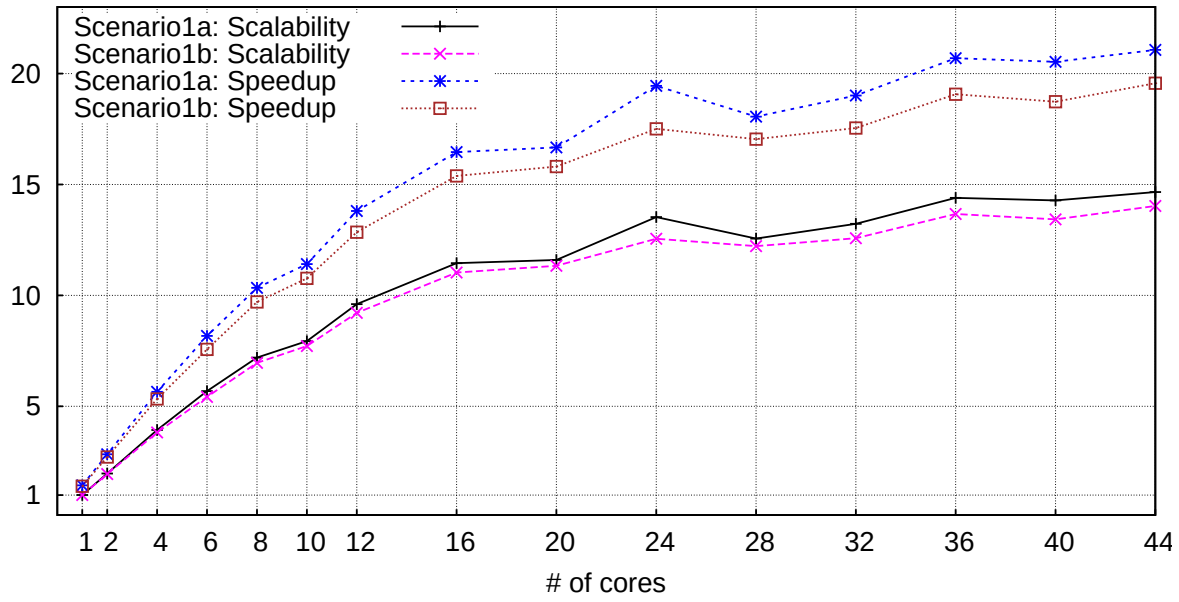


Figure 5.11: Nordic variant 1, Scenario 1: Speedup and scalability computed with Config. II

term stable in both scenarios. However, the final values in Scenario 1b are higher than in 1a. This can be explained from Fig. 5.9, which shows the total active power generated by DGs in all DNs. The disconnection of the DGs in accordance with the LVFRT curves leads to losing approximately 150 MW of distributed generation. This power deficit is covered by importing more power from the TN, thus leading to depressed TN voltages.

Furthermore, Fig. 5.10 shows the voltage at buses E22 and R09 (see Fig. B.1) in two different DNs connected to TN buses 1041 and 4043, respectively. DGs are connected to buses $E22$ and $R09$ and the voltage evolution is compared to the LVFRT curve of Fig. 5.7 to decide whether they will remain connected or not. It can be seen that the DGs in the DN 1041a disconnect at $t \approx 1.2$ s, while those in DN 4043a remain connected. This type of protection schemes (LVFRT) rely on the voltage evolution at DN buses, thus illustrating the necessity for detailed combined dynamic simulations when the DG penetration level becomes significant.

The speedup and scalability for both Scenarios 1a and 1b are shown in Fig. 5.11. Initially, the two-level DDM-based algorithm executed on a single core ($M = 1$) performs similarly to the integrated. When using more computational cores, the proposed algorithm offers a speedup of up to 22 times and the system is simulated in approximately 16 s. The summary of the speedups achieved is given in Tables 5.2 and 5.3.

As regards the scalability of the algorithm, Fig. 5.11 shows that the DDM-based parallel algorithm executes up to 15 times faster in parallel compared to its own sequential execution. Scenario 1a scales slightly better than 1b due to the higher dynamic activity caused by the LVFRT tripping, which disturbs the system and initiates more matrix updates. From the same figure, it can be seen that the parallel algorithm is more efficient in the range of up to 24

Table 5.2: Nordic variant 1, Scenario 1a: Execution times

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)
Integrated (T_1^*)	355 / -	- / -
Config. I	301 / 1.2	26 / 13.7
Config. II	247 / 1.4	16 / 22.2

Table 5.3: Nordic variant 1, Scenario 1b: Execution times

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)
Integrated (T_1^*)	364 / -	- / -
Config. I	323 / 1.1	28 / 13
Config. II	261 / 1.4	19 / 19.2

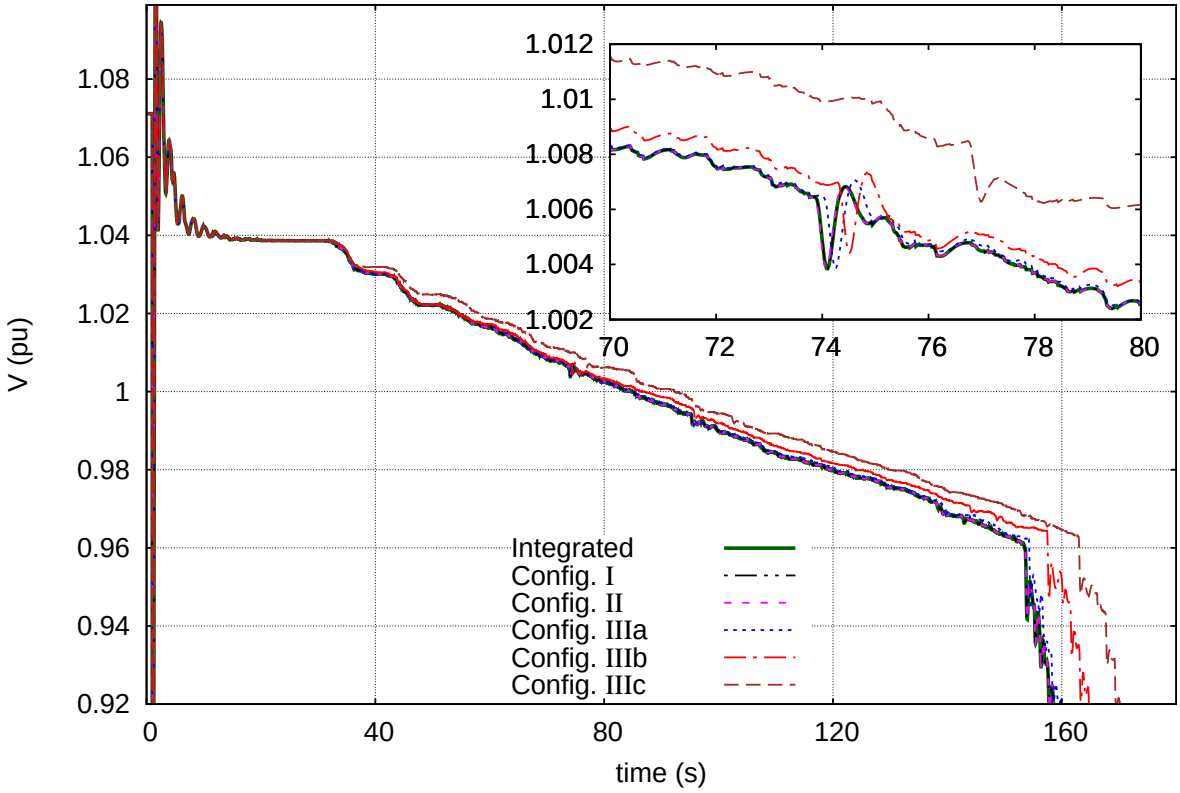


Figure 5.12: Nordic variant 1, Scenario 2a: Voltage evolution at TN bus 4044

cores, while after that the benefit becomes marginal.

5.8.1.2 Scenario 2: Long-term stability study

In this section, a long-term stability study is reported. The disturbance considered is a 5-cycle, three-phase, solid fault near the TN bus 4032, cleared by the opening the faulted line 4032-4042, which remains open. The system is simulated for 240 s with a time-step size of

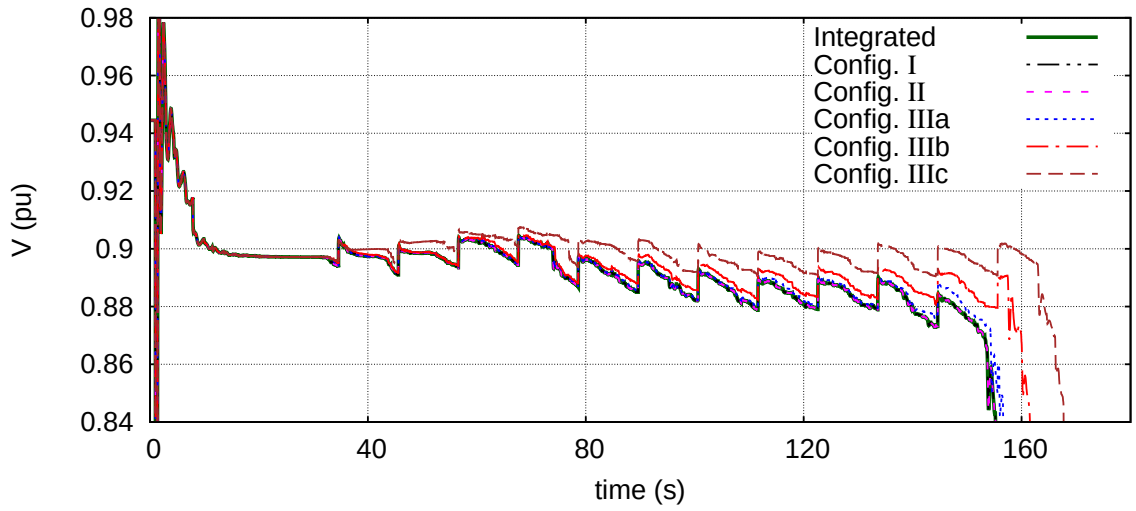


Figure 5.13: Nordic variant 1, Scenario 2a: Voltage evolution at DN bus $1041a - E22$

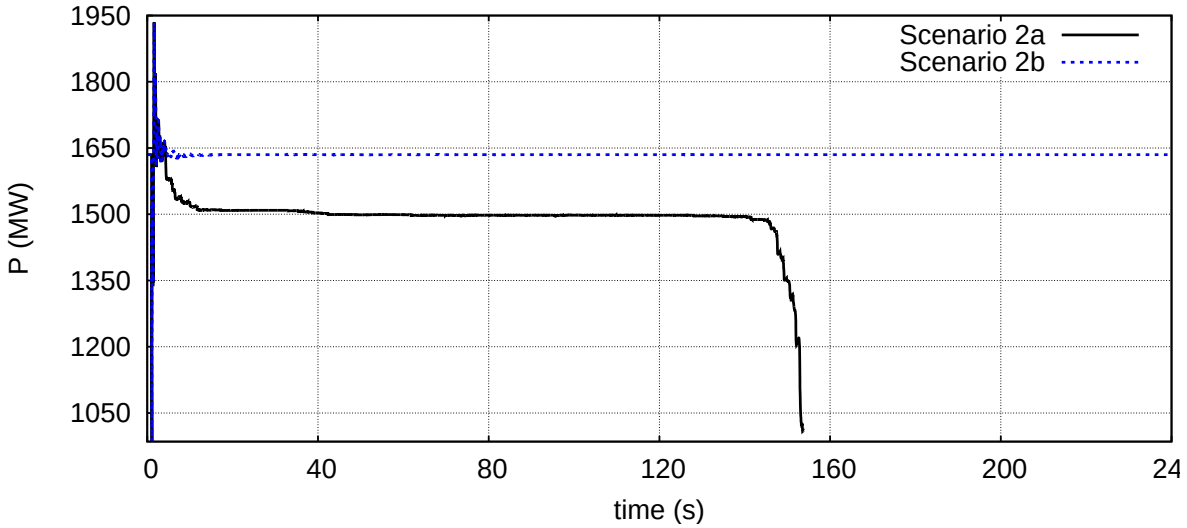


Figure 5.14: Nordic variant 1, Scenario 2: Total active power generation by DGs (Config. II)

one cycle (20 ms).

Scenario 2a is long-term voltage unstable. After the electromechanical oscillations have died out, the system settles at a short-term equilibrium. Then, in the long-term, the system evolves under the effect of Load Tap Changers (LTCs) and the synchronous generator OvereXcitation Limiters (OXLs). Figures 5.12 and 5.13 show the voltage evolution at a TN and a DN bus, respectively.

Figure 5.14 shows the total active power generated by DGs in all the DNs. It can be seen that the DG tripping leads to losing approximately 140 MW in a short period after the fault. This causes the DNs to import the lost power from the TN, thus increasing the TN-DN power transfer and depressing TN voltages. In the long-term, the LTCs act to restore distribution voltages and, hence, the consumption of voltage sensitive loads in DNs. This leads to further

Table 5.4: Nordic variant 1, Scenario 2a: Execution times and inaccuracy of simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	1132 / -	- / -	-
Config. I	1212 / 0.9	98 / 11.6	0.0
Config. II	857 / 1.3	62 / 18.3	0.0
Config. IIIa	758 / 1.5	62 / 18.3	0.004
Config. IIIb	666 / 1.7	58 / 19.5	0.010
Config. IIIc	408 / 2.8	48 / 23.6	0.015

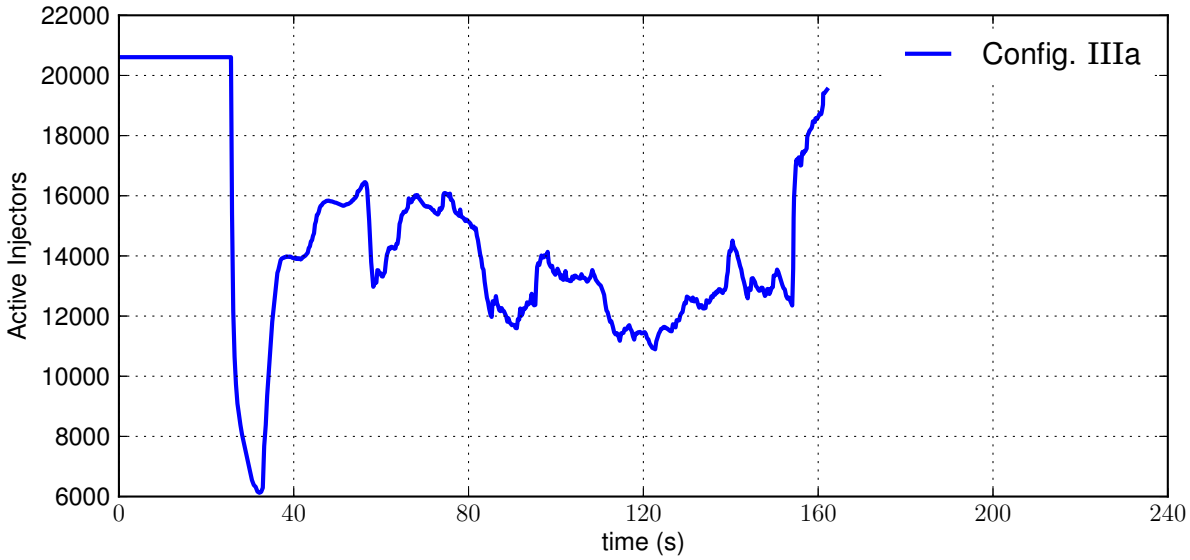


Figure 5.15: Nordic variant 1, Scenario 2a: Number of active injectors

voltage depression at the TN level until the system collapses at $t \approx 155$ s.

Configurations I and II yield the same response as the integrated. Configuration IIIa shows very little error while Configs. IIIb and IIIc cause a shift of the collapsing time by approximately 5 and 10 s, respectively. Nevertheless, the final outcome of the simulation and the collapse mechanisms are simulated correctly.

Table 5.4 shows the simulation time, the speedup (computed using Eq. 2.2) and the maximum inaccuracy over all bus voltages, compared to the integrated method. For the latter computation, only the simulation time until $t \approx 150$ s was considered. In the remaining, the error concerned refers to the delayed simulated voltage collapse.

From the sequential execution timings, it can be seen that Config. I is slower than the integrated. This is due to the extra OHC relating to the two-level partition management, the formulation of the reduced systems, the computing of Schur-complement terms, and the overall higher complexity of the DDM compared to the integrated. On the contrary, Config. II is faster than the integrated due to the localization techniques used. Last, while Config. IIIa is faster than the integrated, it does not offer much higher performance than Config. II. As

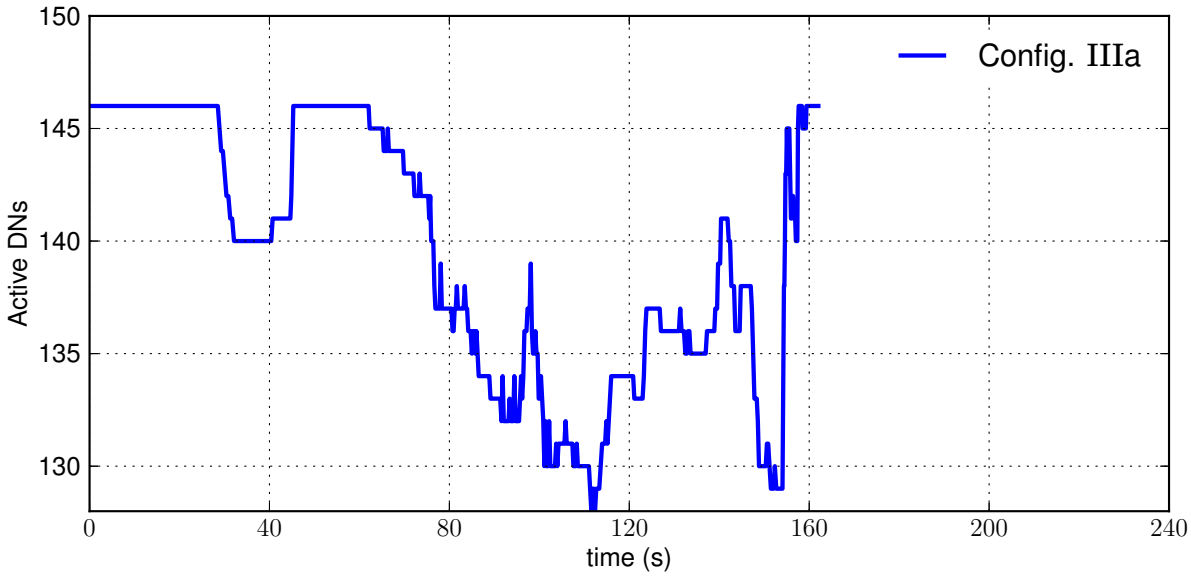


Figure 5.16: Nordic variant 1, Scenario 2a: Number of active DNs

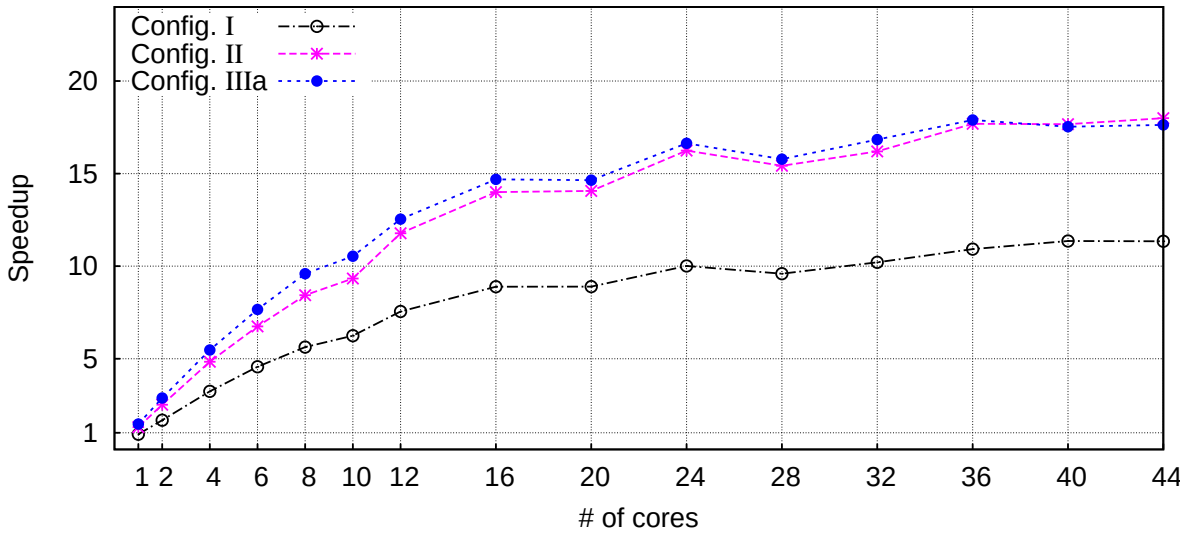


Figure 5.17: Nordic variant 1, Scenario 2a: Speedup computed with Eq. 2.2

explained in Section 4.9.1.1, collapsing scenarios do not exhibit low enough dynamic activity for latency to provide large acceleration.

The performance benefits obtained by latency in this scenario are due to the long-term nature of the voltage collapse allowing the injectors and DNs to become latent *between successive LTC actions*. Figures 5.15 and 5.16 show respectively the number of active injectors and DNs throughout the simulation for Config. IIIa. It can be seen that very few DNs become latent, and towards the end of the simulation all injectors and DNs become active.

From the parallel execution timings in Table 5.4, a speedup of 18.3 times can be obtained while retaining full accuracy of the simulation. A more detailed view is offered in Fig. 5.17, where the speedup is shown as a function of the number of cores used. Furthermore, it can

Table 5.5: Nordic variant 1, Scenario 2a: Time profiling of sequential execution ($M = 1$)

	% of execution time		
	Config. I	Config. II	Config. IIIa
BLOCK A	20	6.49	7.38
BLOCK B	4.6	1.58	4.01
BLOCK C	0.03	0.03	0.04
BLOCK D	3.96	4.32	4.84
BLOCK E	59.73	71.61	63.87
BLOCK F	3.24	6	7.18
Remaining parallel	2.34	3.32	5.08
Remaining sequential	6.1	6.65	7.6
T_P (100-T_S)	93.87	93.32	92.36
T_S (BLOCK C+Rem. Seq.)	6.13	6.68	7.64

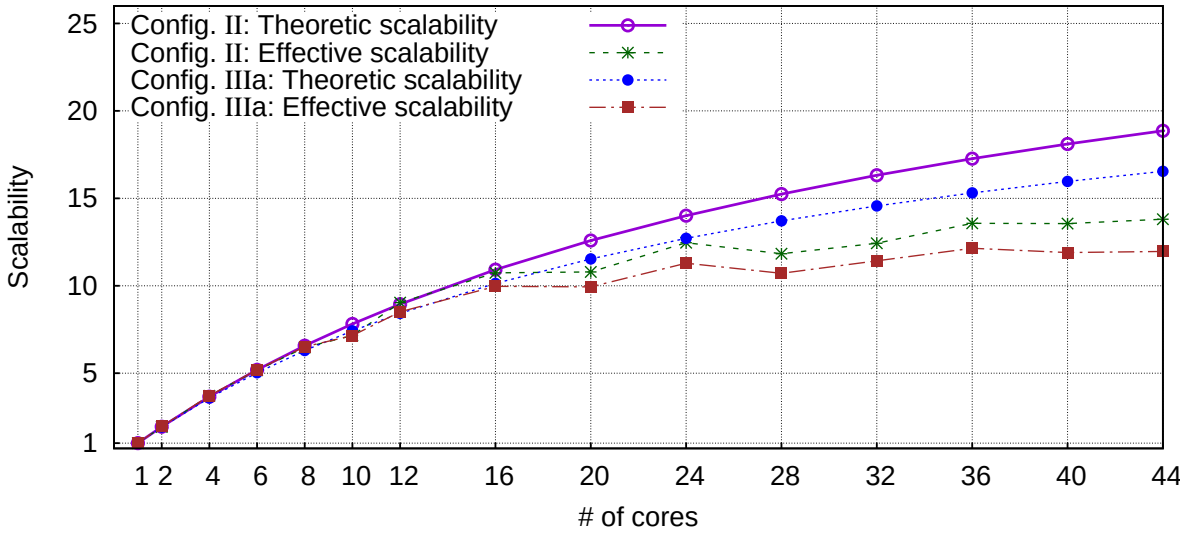


Figure 5.18: Nordic variant 1, Scenario 2a: Effective VS Theoretic scalability

be seen that in parallel execution, Config. IIIa offers no further gain compared to Config. II. Even more, with $M \geq 40$ cores, Config. II becomes faster than IIIa.

Table 5.5 shows the time profiling of the sequential execution ($M = 1$), with the percentage of time spent in each block of the algorithm in Fig. 5.3. As expected, using Config. III leads to lower percentage of parallel work. Using the timings of T_P and T_S, Fig. 5.18 shows the effective scalability (computed with Eq. 2.1) against the theoretic scalability (computed with Eq. 2.5). The difference between them is due to the OHC of the implementation. From Figs. 5.17 and 5.18 it can be seen that the efficiency of the parallelization is very good until $M = 16$, while for higher number of threads the incremental gain is smaller. Nevertheless, the best results are acquired with $M = 44$ and are the ones shown in the parallel execution column of Table 5.4.

Contrary to Scenario 2a, Scenario 2b is long-term stable. In this (very optimistic) scenario, the DGs remain connected to the DNs throughout the simulation, thus supporting the system and avoiding the voltage collapse. Figures 5.19 and 5.20 show the voltage evolution

at a TN and DN bus, respectively. The jumps in Fig. 5.20 correspond to LTC moves in the DN of concern (the shown voltage is not the one controlled by the LTC though).

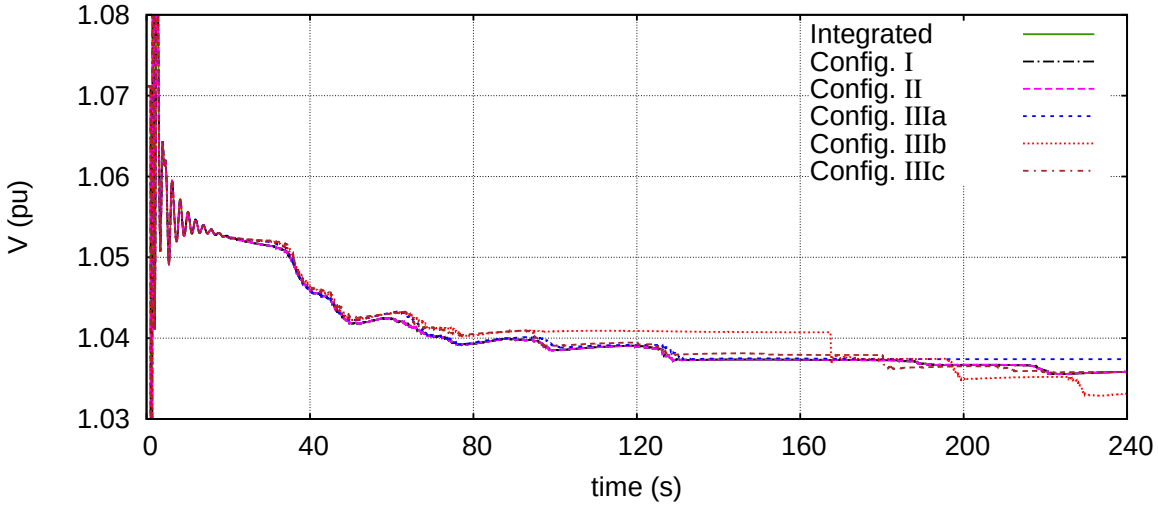


Figure 5.19: Nordic variant 1, Scenario 2b: Voltage evolution at TN bus 4044

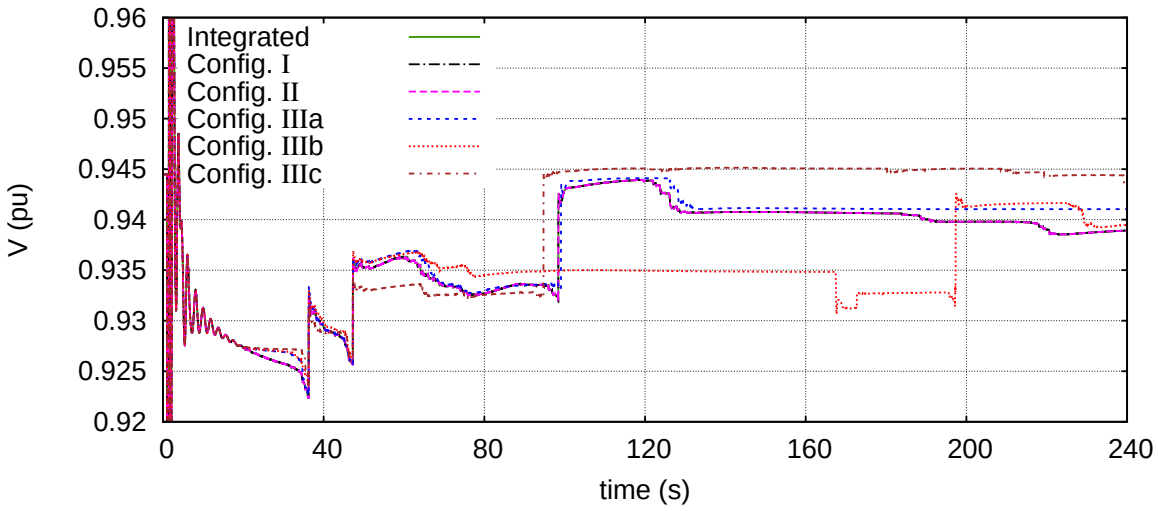


Figure 5.20: Nordic variant 1, Scenario 2b: Voltage evolution at DN bus $1041a - E22$

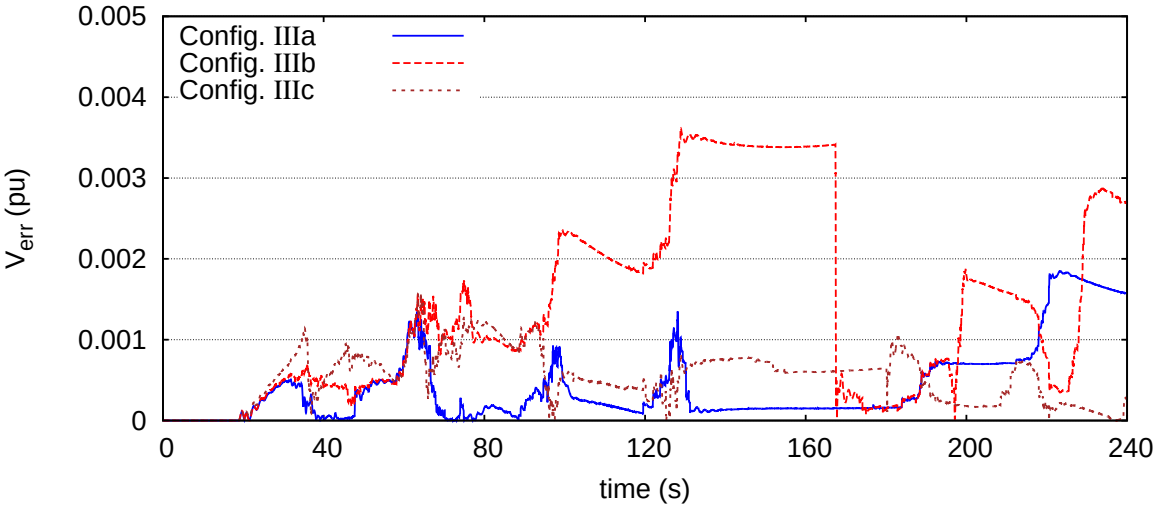


Figure 5.21: Nordic variant 1, Scenario 2b: Absolute voltage error on voltage at TN bus 4044

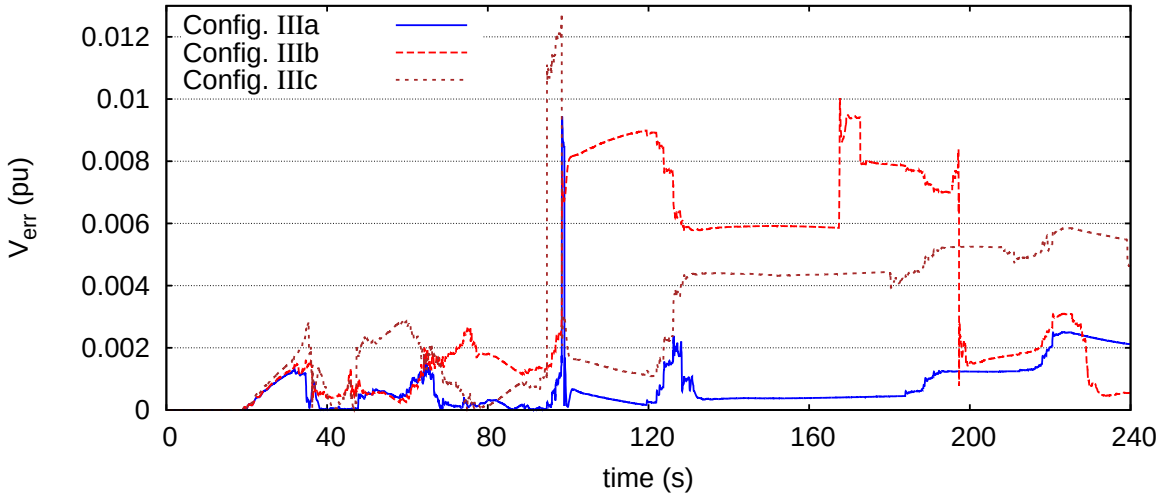


Figure 5.22: Nordic variant 1, Scenario 2b: Absolute voltage error on voltage at DN bus $1041a - E22$

Table 5.6: Nordic variant 1, Scenario 2b: Execution times and inaccuracy of simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	1157 / -	- / -	-
Config. I	1277 / 0.9	103 / 11.2	0.0
Config. II	1095 / 1.1	76 / 15.2	0.0
Config. IIIa	400 / 2.9	43 / 26.9	0.010
Config. IIIb	331 / 3.5	40 / 28.9	0.010
Config. IIIc	253 / 4.6	32 / 36.2	0.014

Their corresponding voltage errors are shown in Figs. 5.21 and 5.22. It can be seen that the main source of inaccuracy is the shift of LTC actions in the system. However, even more than with the ASRT events in the HQ system (see Section 4.9.2), such delays are not considered significant as long as the final voltages are acceptable.

Table 5.6 shows the simulation time, speedup, and maximum inaccuracy over all the bus voltages compared to the integrated method. From the sequential execution timings, it can be seen that Config. I is slower than the integrated as in Scenario 2a. Configuration II offers some small speedup, while Config. III provides a higher speedup in sequential execution while also introducing some error.

Figure 5.23 shows the number of active injectors and DNs throughout the simulation for Config. IIIa. It can be seen that in the short-term, until the post-fault electromechanical oscillations die out, the injectors and DNs remain active. On the other hand, in the long-term many injectors and DNs exhibit low dynamic activity, and towards the end of the simulation almost all of them become latent. This verifies the observation that *in the short-term, when*

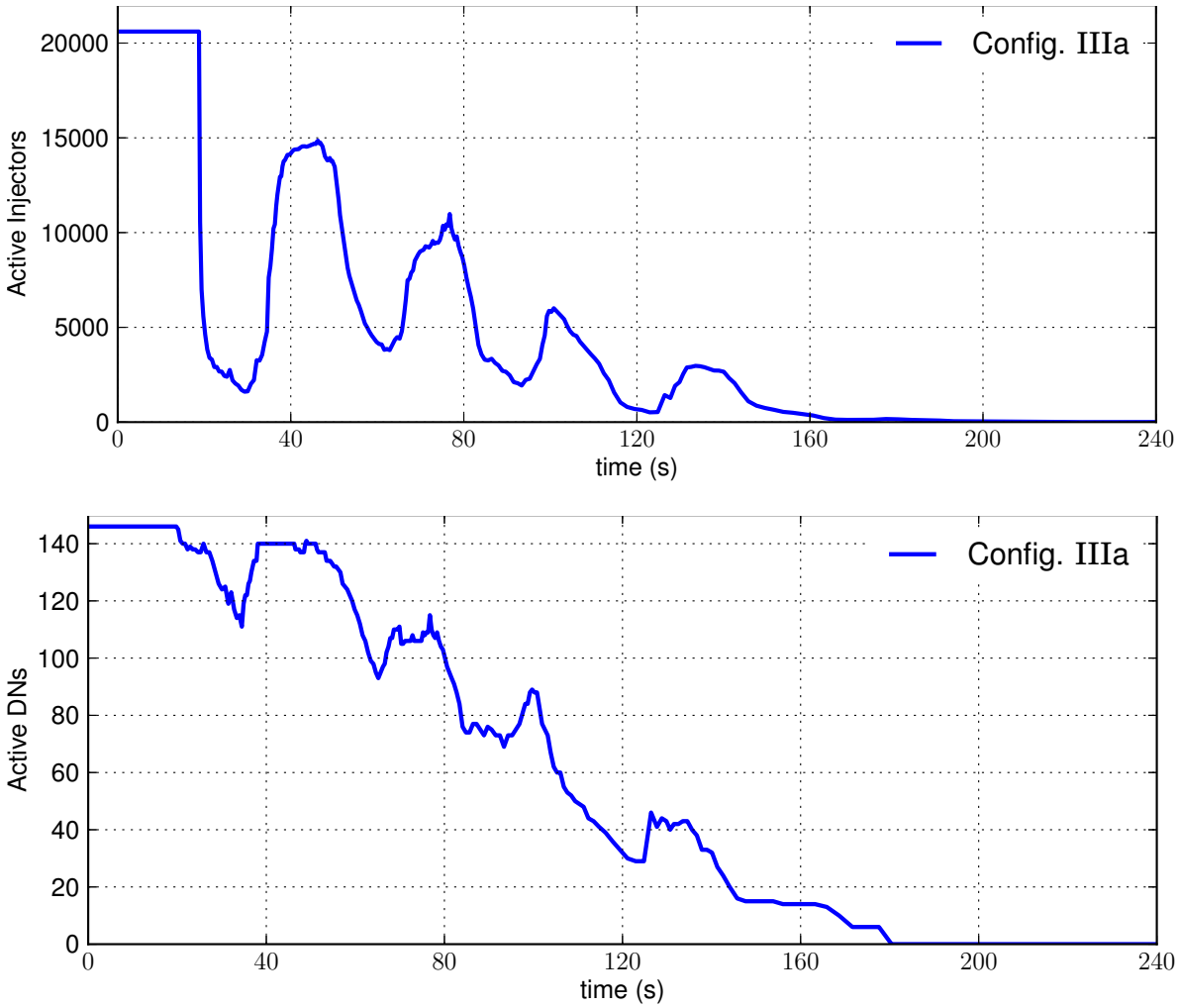


Figure 5.23: Nordic variant 1, Scenario 2b: Number of active injectors and DNs

there is high dynamic activity in the system with frequent matrix updates, parallelization is the main source of speedup. In the long-term, when dynamics with larger time constants dominant, latency is the main source of speedup.

From the parallel execution timings of Table 5.6, it can be seen that all configurations offer significant speedup when parallelized: up to 15.2 times without any inaccuracy and up to 28.9 times when latency is used and $\epsilon_L \leq 0.2$ MVA.

Table 5.7 shows the time profiling of the sequential execution ($M = 1$), with the percentage of time spent in each block of the algorithm in Fig. 5.3. As expected, using Config. III leads to an overall lower percentage of parallel work. Figure 5.24 shows the speedup as a function of the number of cores used; while, Fig. 5.25 shows the theoretic (using the timings of T_P and T_S) and the effective scalability. From these figures, it can be seen that the efficiency of the parallelization is very good until $M = 24$, while for a higher number of threads the incremental gain is small.

Table 5.7: Nordic variant 1, Scenario 2b: Time profiling of sequential execution ($M = 1$)

	% of execution time		
	Config. I	Config. II	Config. IIIa
BLOCK A	12.35	5.38	8.39
BLOCK B	2.28	3.41	3.18
BLOCK C	0.03	0.02	0.05
BLOCK D	6.2	4.96	4.59
BLOCK E	69.21	72.7	62.78
BLOCK F	3.63	4.75	6.98
Remaining parallel	3.52	7.78	8
Remaining sequential	2.78	3	6.03
T_P (100-T_S)	97.19	96.98	93.92
T_S (BLOCK C+Rem. Seq.)	2.81	3.02	6.08

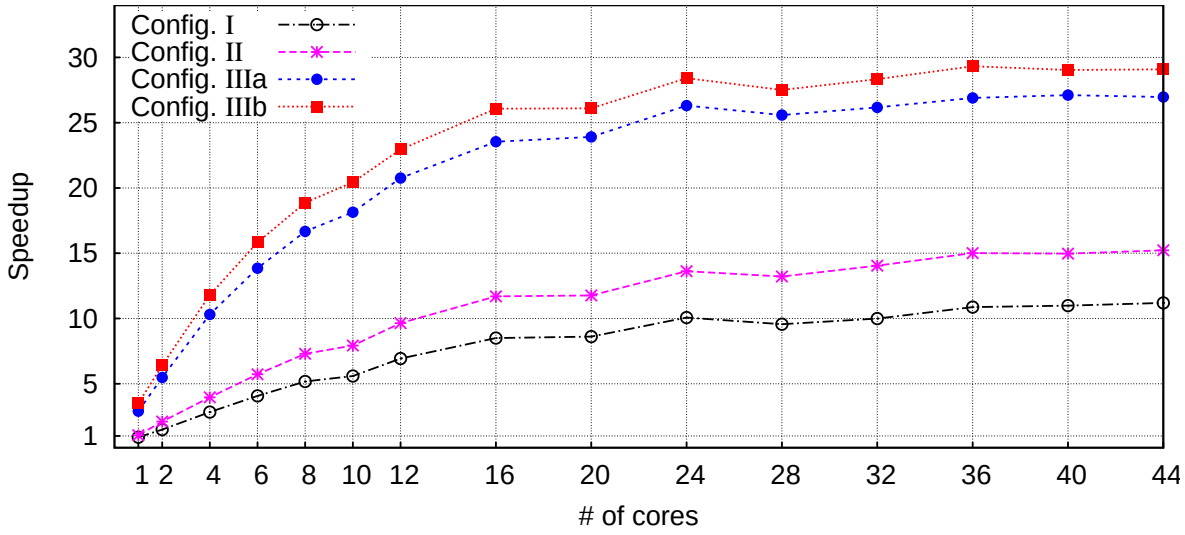


Figure 5.24: Nordic variant 1, Scenario 2b: Speedup computed with Eq. 2.2

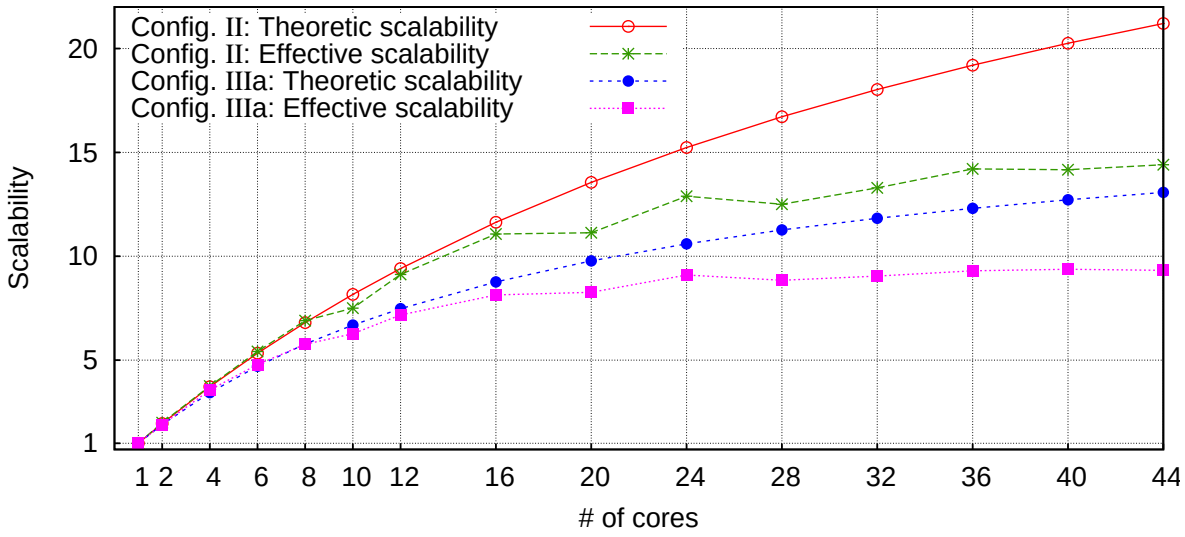


Figure 5.25: Nordic variant 1, Scenario 2b: Effective VS Theoretic scalability

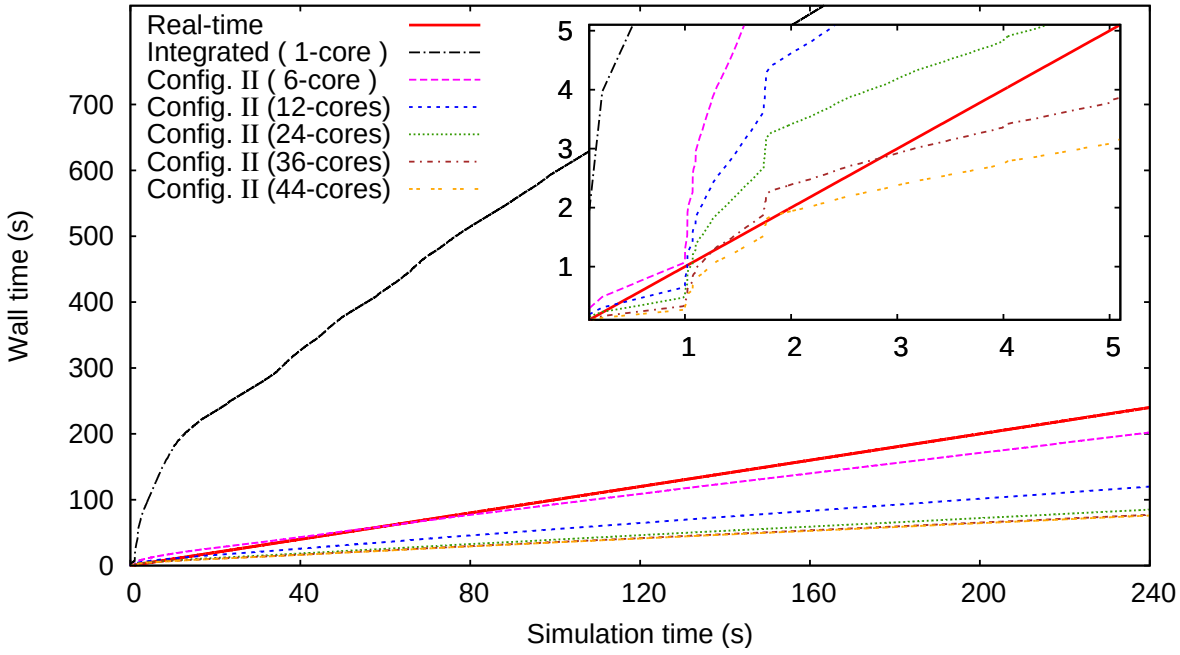


Figure 5.26: Nordic variant 1, Scenario 2b: Real-time performance of algorithm

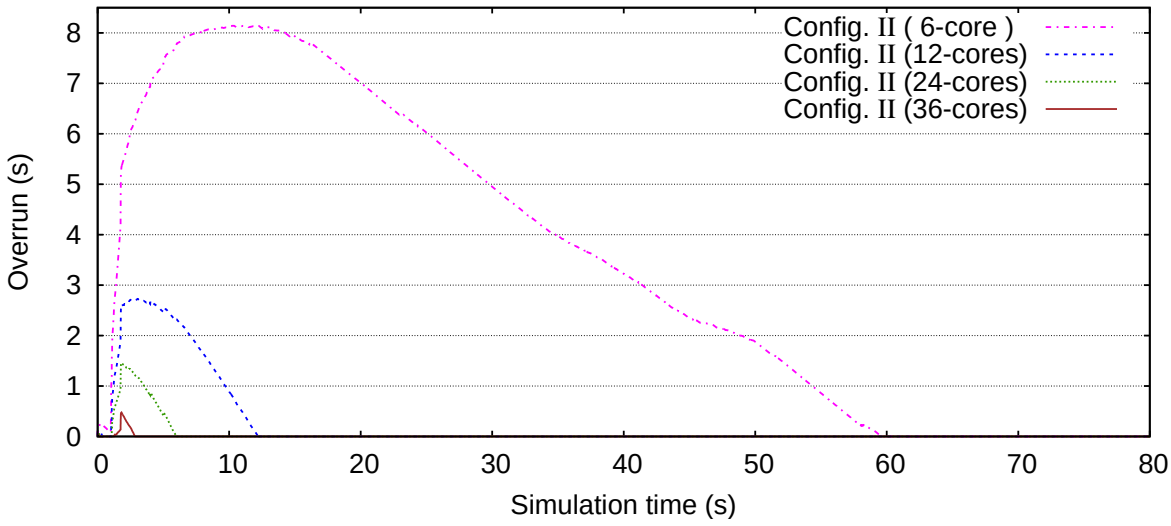


Figure 5.27: Nordic variant 1, Scenario 2b: Overrun of simulations

Finally, Fig. 5.26 shows the real-time performance of the algorithm with Config. II. On this 15000-bus T&D power system, the two-level DDM-based algorithm performs faster than real-time when executed on 44 cores. With fewer cores, some overrun is observed as shown in Fig. 5.27.

5.8.2 Nordic variant 2 system

This is the second combined T&D system, also based on the Nordic system, this time expanded with 40 DNs in the Central area. In total, the system includes 3108 buses, 20 large

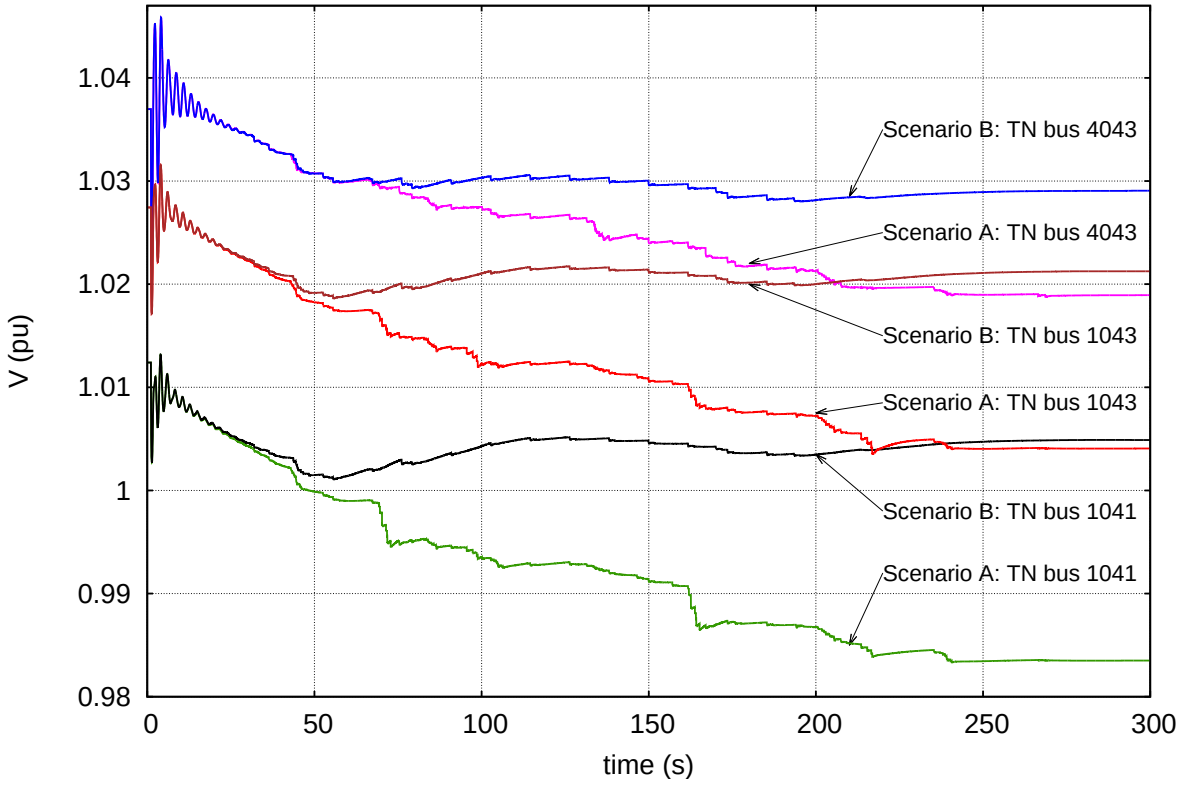


Figure 5.28: Nordic variant 2 (Config. II): Voltage evolution at three TN buses

and 520 small synchronous generators, 600 induction motors, 360 type-3 WTs, 2136 voltage-dependent loads, and 56 LTC-equipped transformers. The resulting DAE model has 36504 differential-algebraic states. A more detailed system description can be found in Section 1.3 and its one-line diagram in Fig. B.5.

First the system is decomposed on the boundary between TN and DNs, thus creating the Central sub-domain with the TN and $L = 40$ Satellite sub-domains with the DNs. Next, each sub-domain is decomposed into its network and injectors. More specifically, $N_{S1} = N_{S2} = ... = 90$ and $N_C = 36$.

A long-term scenario is considered for this system involving the outage of transmission line $4061 - 4062$ in the South area (see Fig. B.5). The system is simulated for 300 s with a time-step size of one cycle (20 ms). Although large, this disturbance leads to a stable response. Furthermore, this contingency is simulated twice. In Scenario A, the DNs are passive and the reactive power set-points of the DGs remain constant throughout the simulation. On the contrary, in Scenario B, each DN is equipped with a Distribution Network Voltage (DNV) controller as detailed in [VV13]. The latter is centralized at the DN level. It is based on Model Predictive Control (MPC) to keep the voltages of some DN buses within desired limit, by changing the DG reactive powers. Each DN is equipped with such a controller, but the various controller instances do not exchange any information. The main purpose of this study was to investigate the contribution of DNV controllers to the bulk transmission system

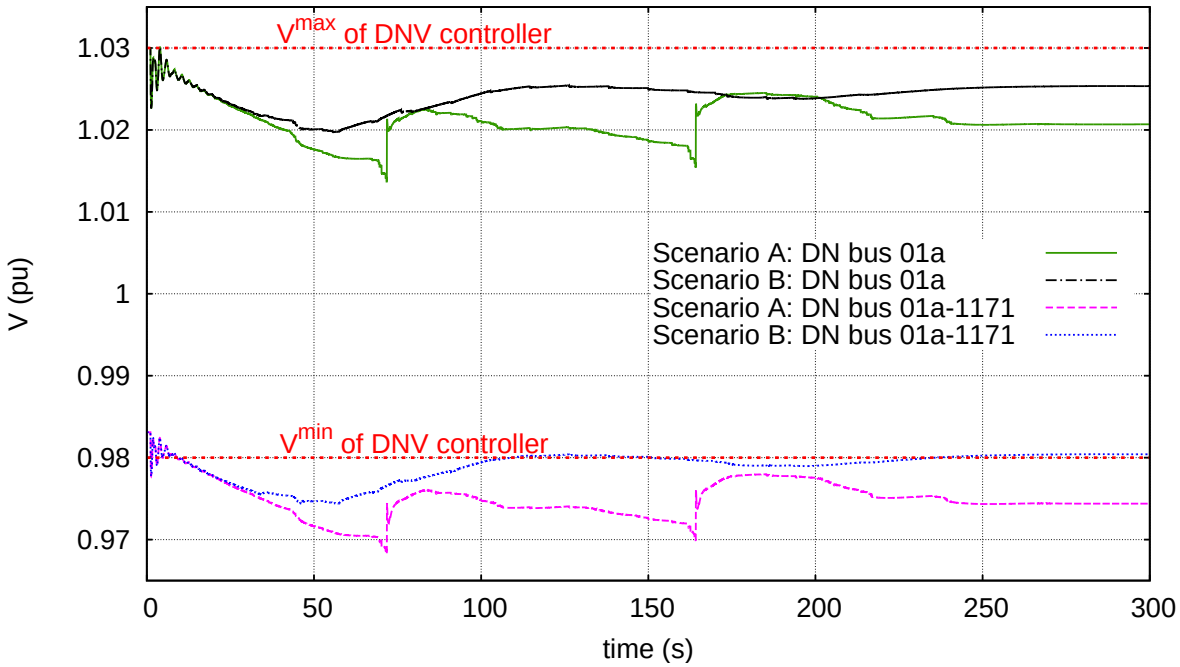


Figure 5.29: Nordic variant 2 (Config. II): Voltage evolution at two DN buses

dynamic behavior.

As with the previous system, different base powers are considered for the Central ($S_{baseC} = 100$ MVA) and Satellite ($S_{baseS} = 2$ MVA) sub-domains. For the integrated method, the smallest one is used for the entire system to ensure the accurate solution of the DN DGs. Moreover, only the fully accurate Configs. I and II have been considered in this study.

In Scenario A (DNV controllers not in service), DG units operate with constant reactive power set-points and do not take part in voltage control. This leaves only the traditional voltage control by LTCs. The long-term evolution of the system, until it returns to steady state, is shown in Figs. 5.28 and 5.29. It is driven by the LTCs, essentially, in response to the voltage drops initiated by the line tripping. Overall, 112 tap take place until a steady-state equilibrium is reached.

Figure 5.28 shows the TN voltage evolution at three representative buses of the Central area. The voltage at bus 1041 is the most impacted but remains above 0.985 pu. All DN voltages are successfully restored in their dead-bands by the LTCs, which corresponds to a stable evolution. For instance, Fig. 5.29 shows the voltage evolution at two DN buses: 01a, controlled by an LTC with a $[1.02\ 1.03]$ pu dead-band, and $01a - 1171$, located further away in the same DN.

In Scenario B, the same disturbance is considered but the DNs equipped with the DNV controllers aimed at keeping the DN voltages between $V_{min} = 0.98$ pu and $V_{max} = 1.03$ pu. This interval encompasses all LTC deadbands, so that there is no conflict between LTC and DNV controllers. Each DNV controller gathers measurements from its DN, solves an MPC problem, and every $8 - 12$ s (randomly selected action interval) adjusts the reactive power

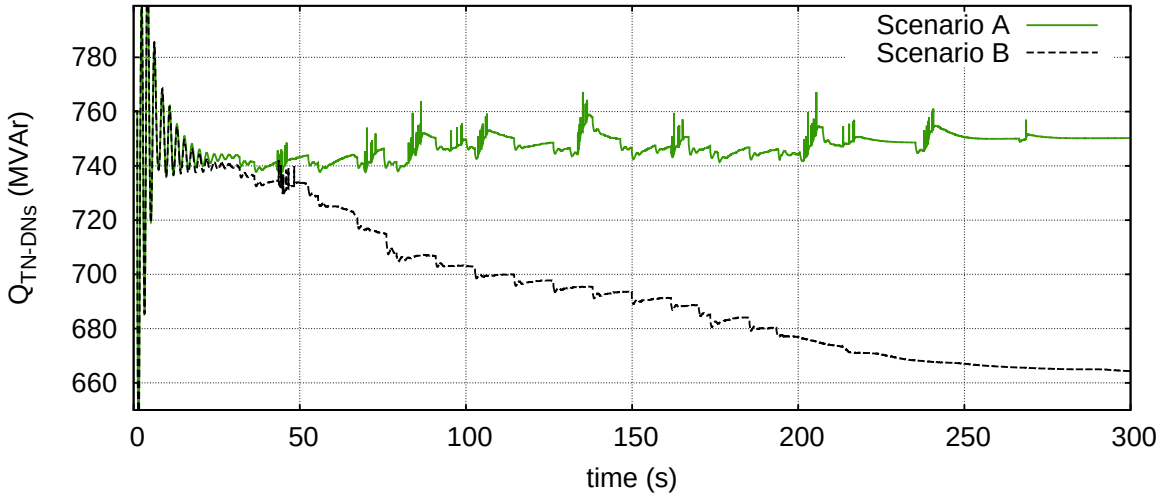


Figure 5.30: Nordic variant 2 (Config. II): Total reactive power transfer from TN to DNs

Table 5.8: Nordic variant 2: Execution times

	Sequential execution time ($M = 1$) (seconds/speedup)		Fastest parallel execution time (seconds/speedup)	
Scenario	A	B	A	B
Integrated (T_1^*)	448 / -	475 / -	- / -	- / -
Config. I	479 / 0.9	475 / 1.0	45 / 10.0	54 / 8.8
Config. II	392 / 1.1	392 / 1.2	40 / 11.2	48 / 9.9

set-points of the DGs.

The resulting voltage evolutions for the same TN and DN buses can be found in Figs. 5.28 and 5.29 for easier comparison. With respect to Scenario A, a steady state is reached at almost the same time, while the TN voltages are slightly higher. The voltages at DN buses are restored above V_{min} by the DNV controller. It is worth mentioning that the number of tap changes has decreased from 112 to 35, showing that the sharing of the control effort by active DNs reduces the wear of LTCs. The DN buses such as $01a - 1171$ in Fig. 5.29 have their voltages increased by the additional reactive power produced by the coordinated DG units. For instance, in Scenario B, the DG participation decreases by almost 90 MVAr the net reactive power load seen by the TN (see Fig. 5.30), which contributes to increasing TN voltages.

Table 5.8 shows the execution times and speedup for both scenarios. In sequential execution, Config. I has similar performance as the integrated and Config. II offers some speedup. In parallel execution, Scenarios A and B exhibit 11.2 and 9.9 times faster execution, respectively. Next, Figs. 5.31 and 5.32 show the speedup and scalability of the two scenarios. Scenario A exhibits slightly better scalability due to its higher dynamic activity; in Scenario B, as DGs support the DN voltages, there are less LTC actions, the voltages reach faster their deadbands and less contribution is provided by the large TN-connected generators.

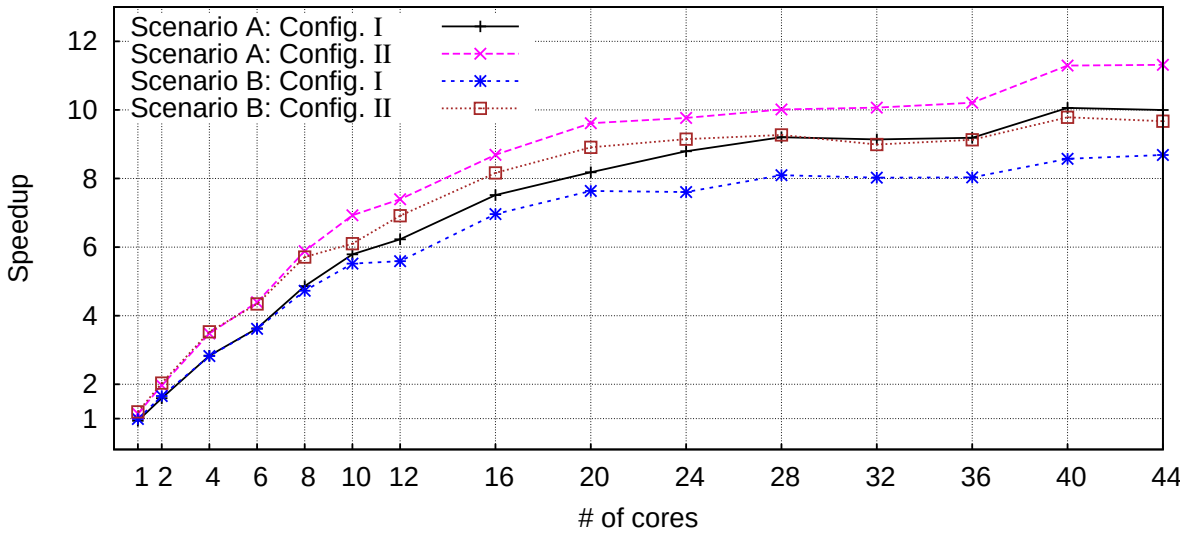


Figure 5.31: Nordic variant 2: Speedup computed with Eq. 2.2

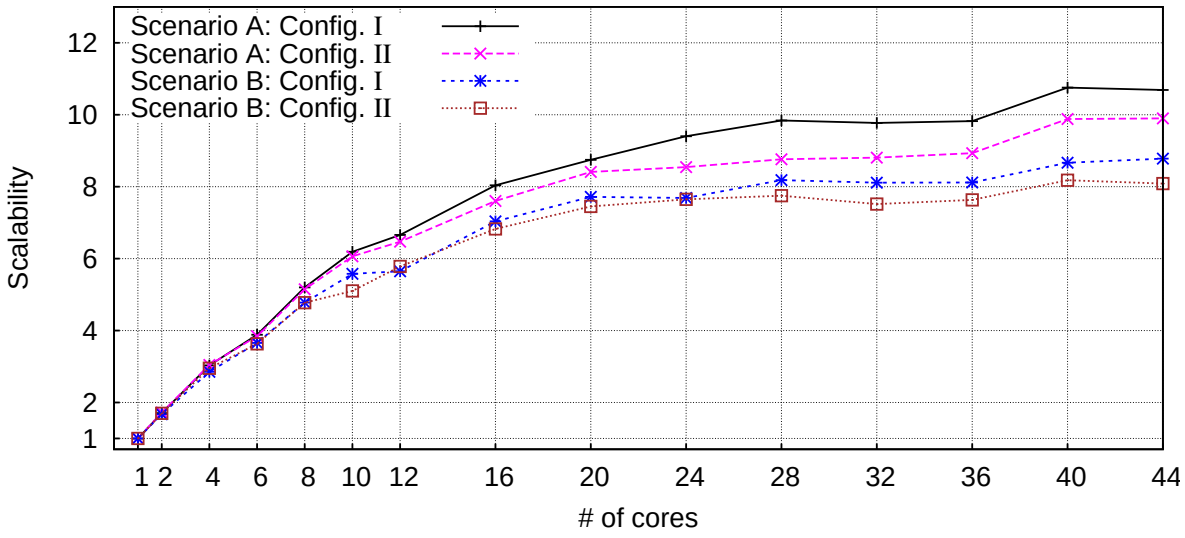


Figure 5.32: Nordic variant 2: Effective scalability with Eq. 2.1

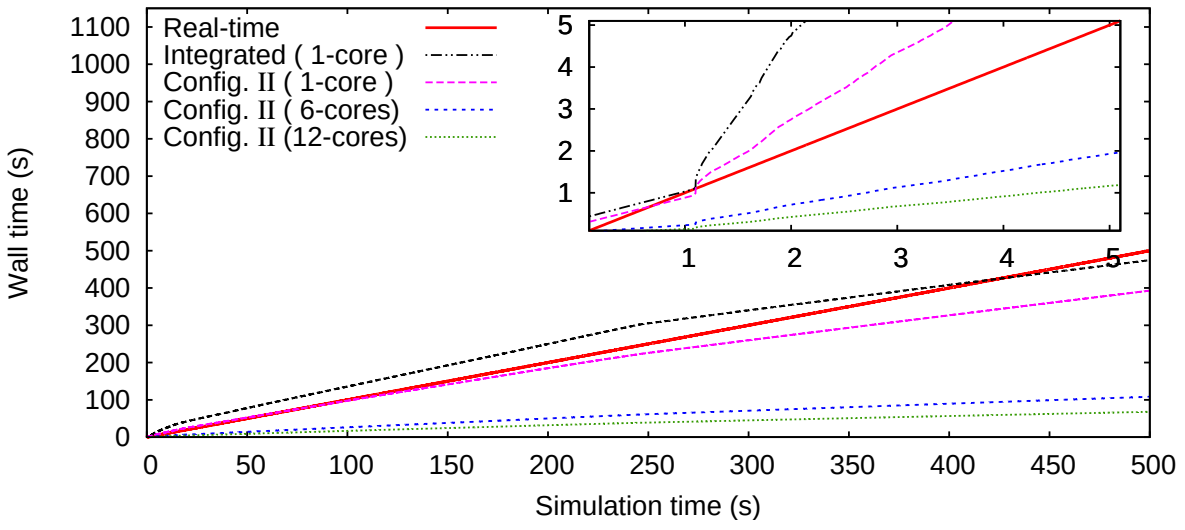


Figure 5.33: Nordic variant 2, Scenario B: Real-time performance of algorithm

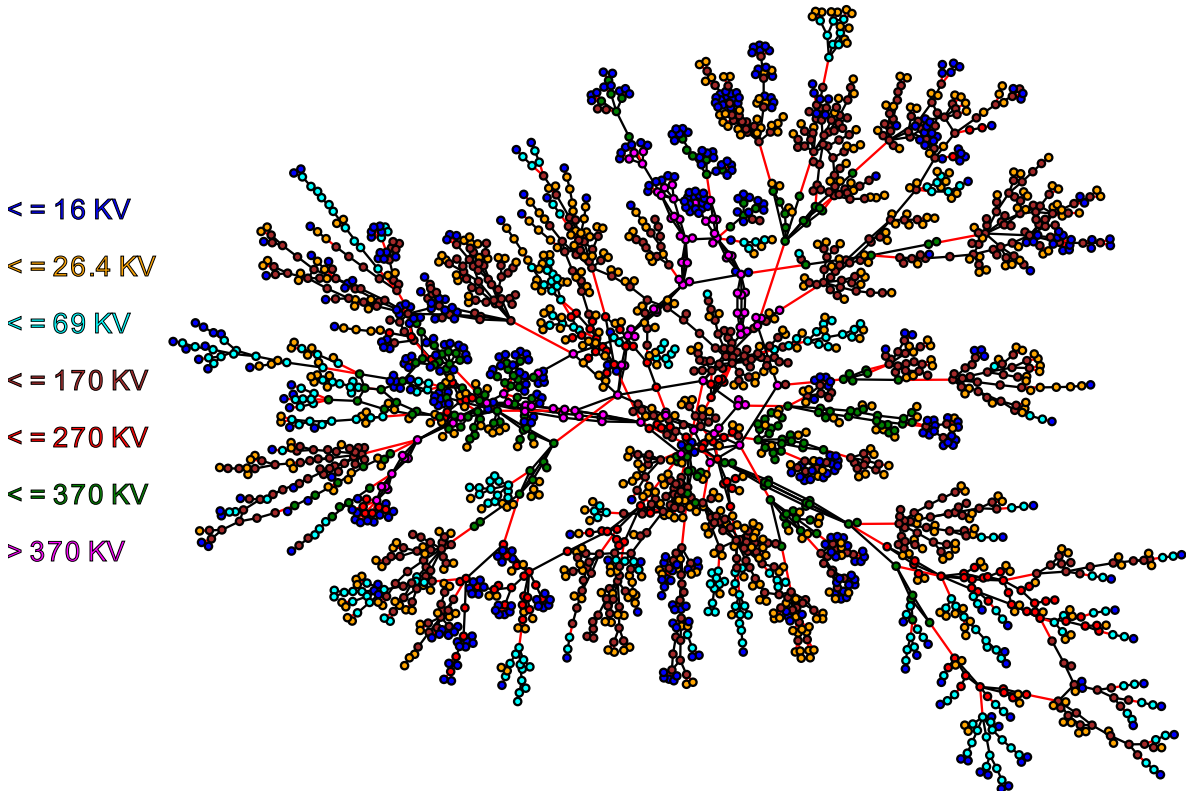


Figure 5.34: HQ: Graph of the entire network (2565 buses)

Finally, Fig. 5.33 shows the real time performance of the algorithm for Scenario B. In this test-case, the faster than real-time performance would allow implementing a controller-in-the-loop structure. That is, the simulator can assume the role of the real system and communicate the DN measurements (with some simulated measurement noise) to the DNV controller implementations. Then, the DG set-points are calculated and communicated back to the simulator. The simulation should be slowed down to match real time in order to test the performance of these controllers.

5.8.3 Hydro-Québec system

In this section, the test-case studied in Section 4.9.2 is revisited with the use of the two-level DDM. Even though this is a TN, its particular structure with radially connected sub-transmission systems[1] allows to exploit the two-level decomposition. An example is shown in Fig. B.7. Algorithm 5.1 can be used to identify a star-shaped decomposition of the system network. Figure 5.34 shows the full network graph; while, Fig. 5.35 shows the graph of the Central sub-domain obtained with Algorithm 5.1. The graph analysis and the partition was performed using NetworkX, a Python software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [HSS08].

[1] sometimes referred to as distribution in Canada

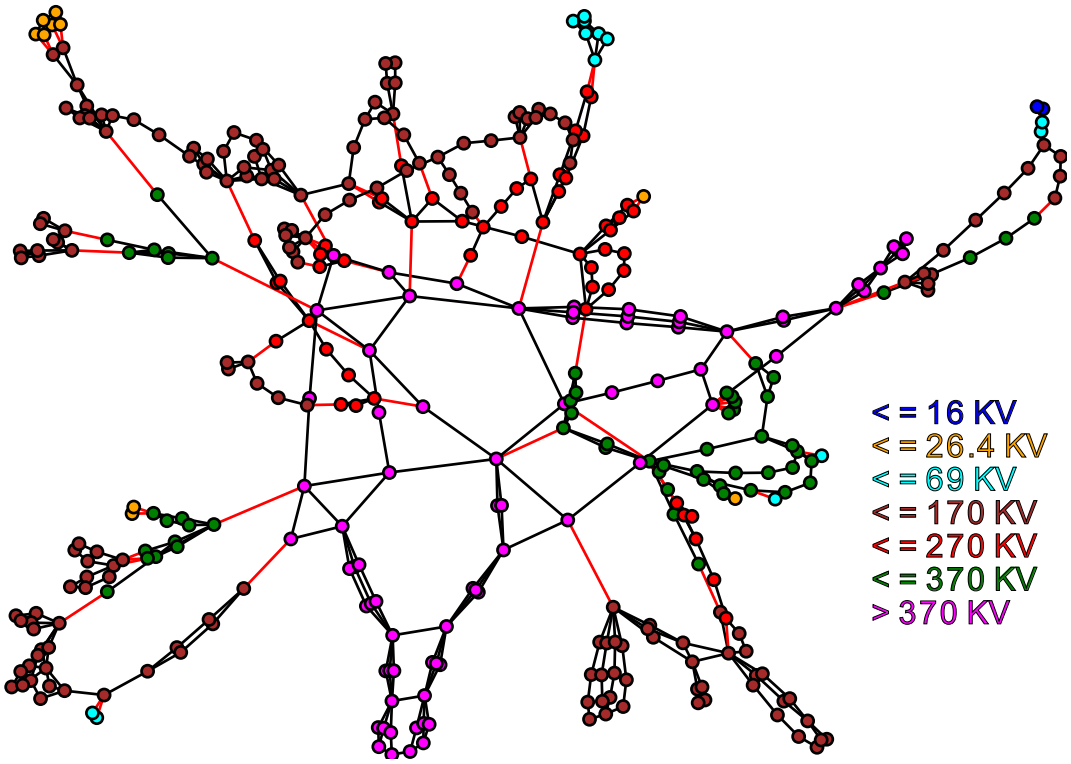


Figure 5.35: HQ: Graph of the Central sub-domain (387 buses) with Satellite sub-domains including more than one bus

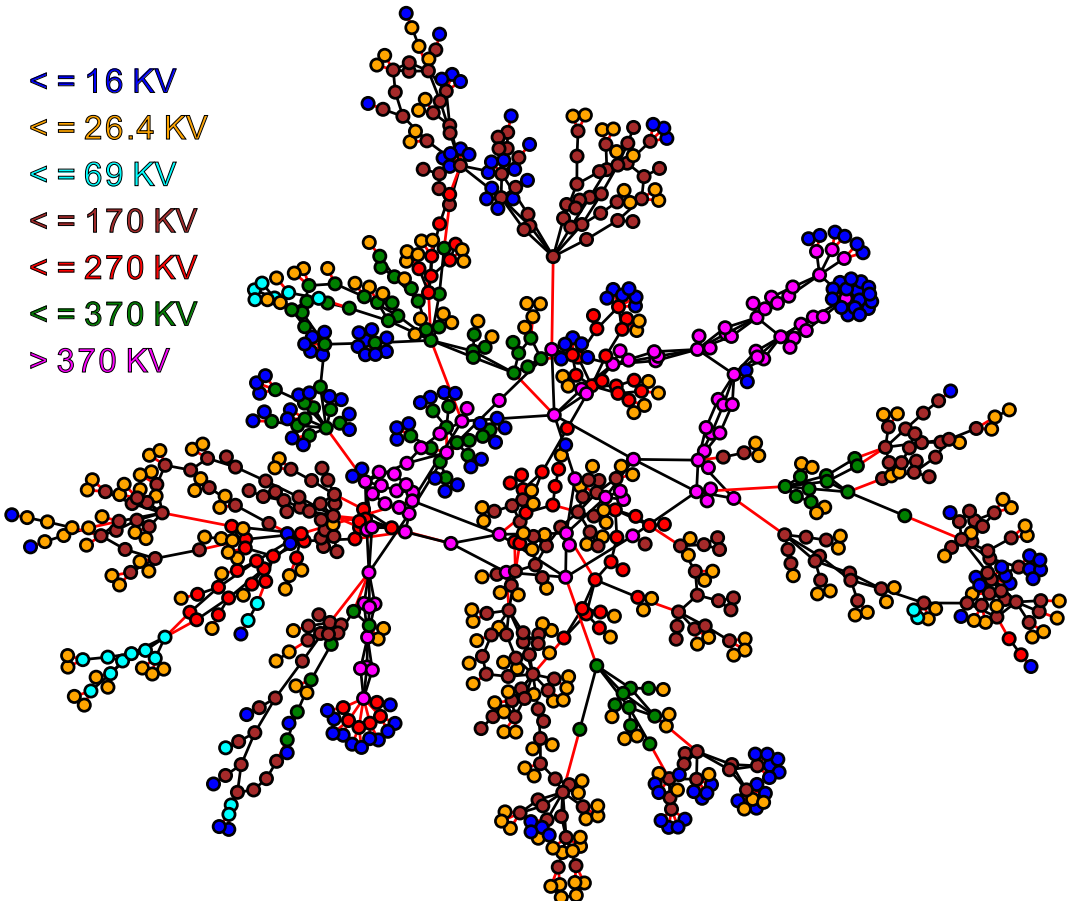


Figure 5.36: HQ: Graph of the Central sub-domain (963 buses) with Satellite sub-domains including more than five buses

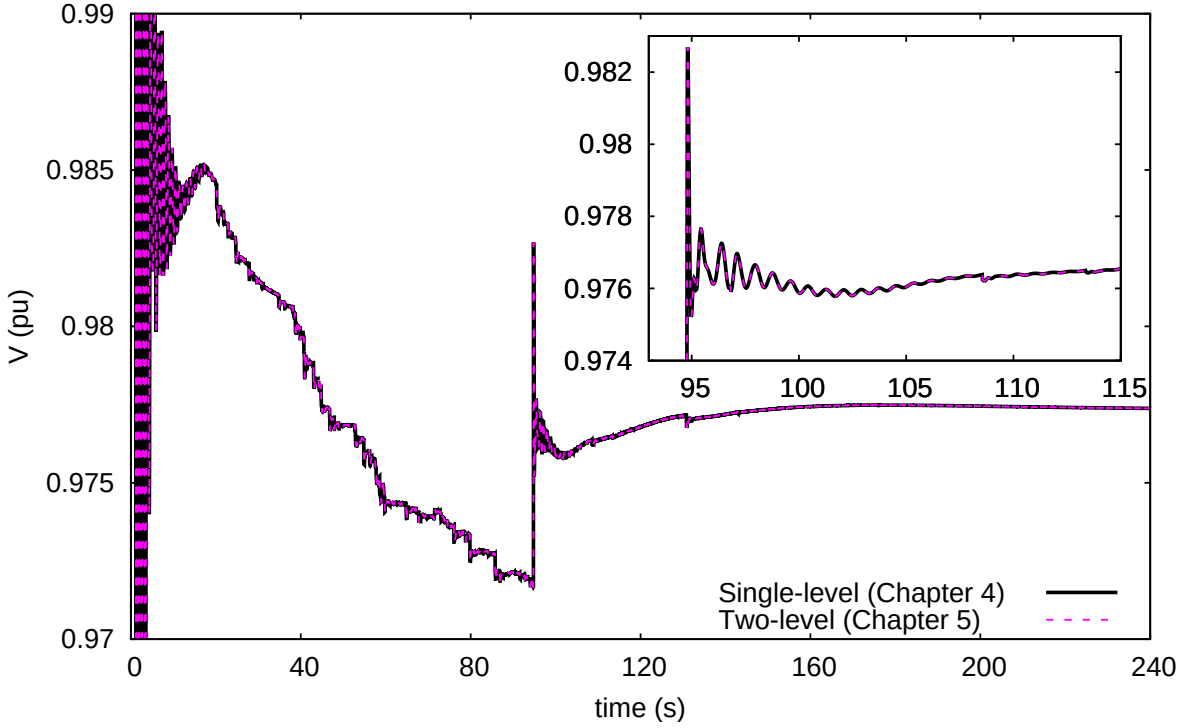


Figure 5.37: HQ: Voltage evolution at bus 702 with the two DDMs (Config. II)

The graph nodes represent the network buses, and their nominal voltages are distinguished by different colors. The graph edges represent network lines (colored black) and transformers (colored red). Thus, the network is split into the Central sub-domain and 450 Satellite sub-domains. With this splitting, the Central sub-domain consists of 387 buses while the remaining 2178 buses are located in the Satellite sub-domains. However, this decomposition leads to many Satellite sub-domains consisting of one or two buses. Such small Satellite sub-domains do not offer sufficient workload and lead to increased imbalance (especially when the option of OpenMP for static scheduling is used) and OHC. Hence, a decomposition with larger Satellite sub-domains has been preferred. Figure 5.36 shows the Central sub-domain if Satellite sub-domains with *at least five buses* are used. This choice leads to 80 Satellite sub-domains with a total of 1602 buses; each with a substantial workload.

Since in this system the Satellite sub-domains are not DNs but part of the decomposed TN, the same base power and latency tolerance is used for all sub-domains. That is $S_{baseC} = S_{baseS} = 100$ MVA and $\epsilon_{LS} = \epsilon_{LC}$ in Config. III. Furthermore, the same contingency considered in Section 4.9.2 is used for comparison reasons.

Figure 5.37 shows the voltage evolution at TN bus 702 with the two algorithms (of Chapters 4 and 5) using Config. II. The dynamic response of the system is exactly the same in both algorithms as they solve the same DAE system without any approximations. Next, Fig. 5.38 shows the voltage evolution at the same bus with the two-level DDM with different configurations. Concerning the dynamic response of the system, the observations made in

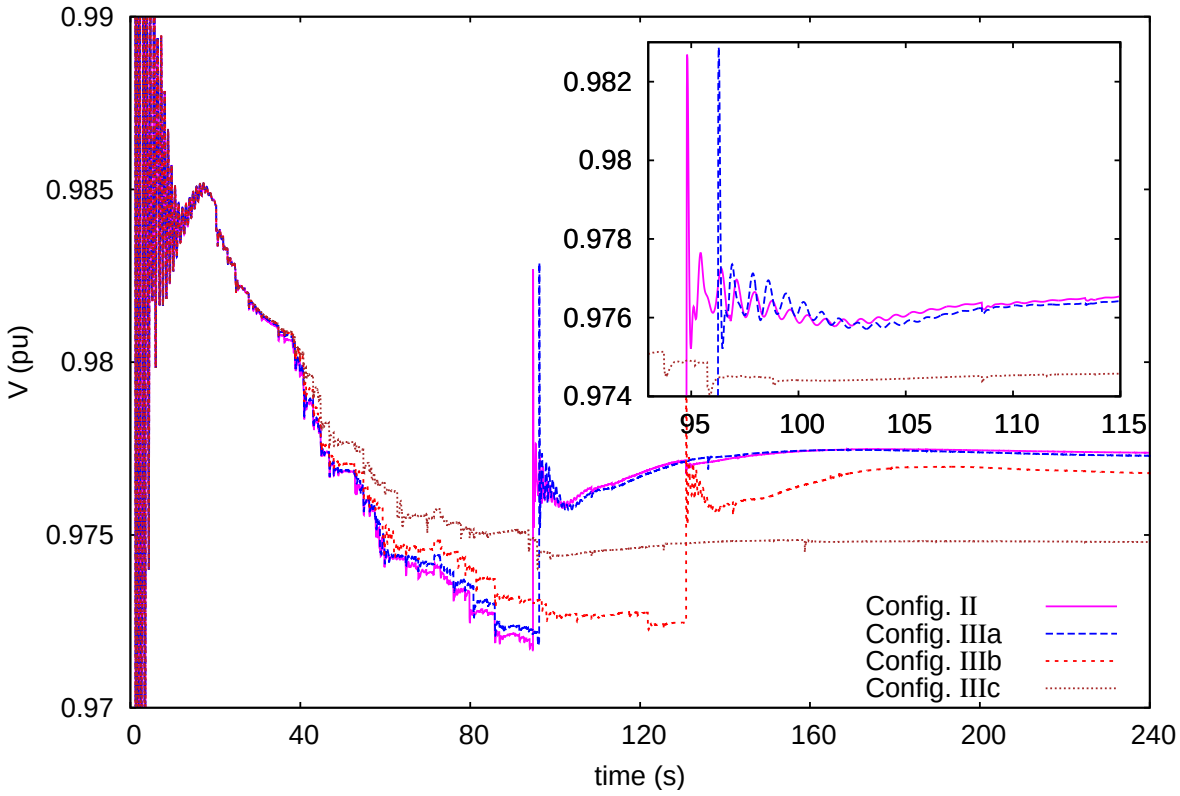


Figure 5.38: HQ: Voltage evolution at bus 702 with the two-level DDM

Table 5.9: HQ: Execution times and inaccuracy of simulation

	Sequential execution time ($M = 1$) (seconds/speedup)	Fastest parallel execution time (seconds/speedup)	Maximum error on voltage (pu)
Integrated (T_1^*)	413.8 / -	- / -	-
Config. I	443.6 / 0.9	53.7 / 7.7	0.0
Config. II	273.2 / 1.5	42.8 / 9.7	0.0
Config. IIIa	134.6 / 3.1	40.8 / 10.1	0.010
Config. IIIb	105.1 / 3.9	35.3 / 11.7	0.012
Config. IIIc	85.6 / 4.8	31.8 / 13.0	0.018

Section 4.9.2 hold true for this algorithm as well.

Table 5.9 shows the simulation time, speedup and maximum inaccuracy over all the bus voltages compared to the integrated method. From the sequential execution timings, it can be seen that Config. I shows some slow-down while the remaining offer various speedups. The highest performance is offered by Config. IIIc while also introducing the largest error. Figure 5.39 shows the absolute voltage error of transmission bus 702. As with the algorithm of Chapter 4, the error peaks at $t = 94$ s due to the shifted ASRT event, whose importance was commented in the previous chapter. A value of $\epsilon_L \leq 0.2$ MVA achieves a good balance between speedup and inaccuracy.

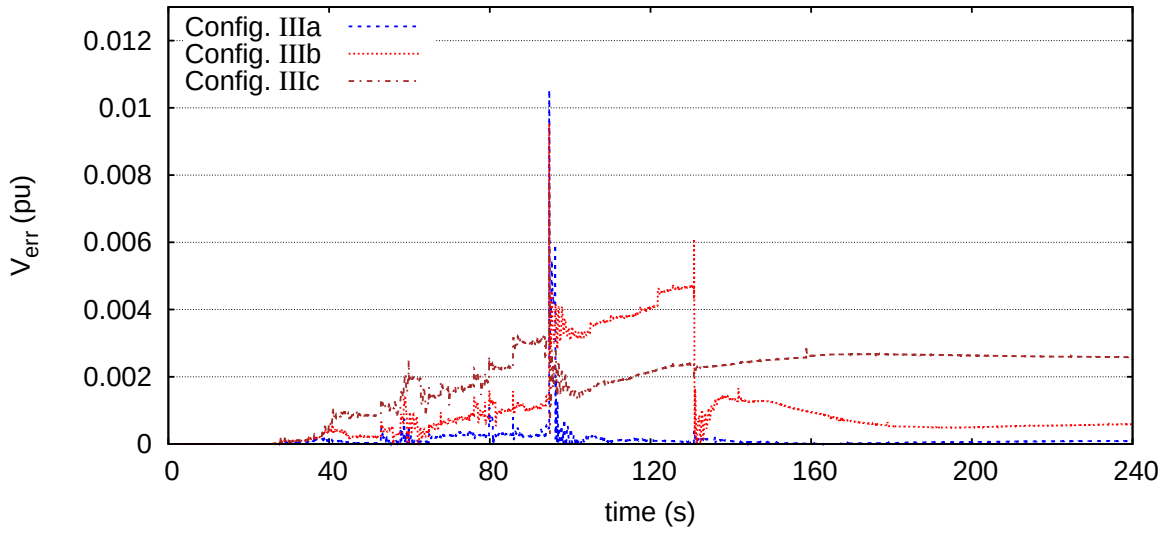


Figure 5.39: HQ: Absolute voltage error on bus 702

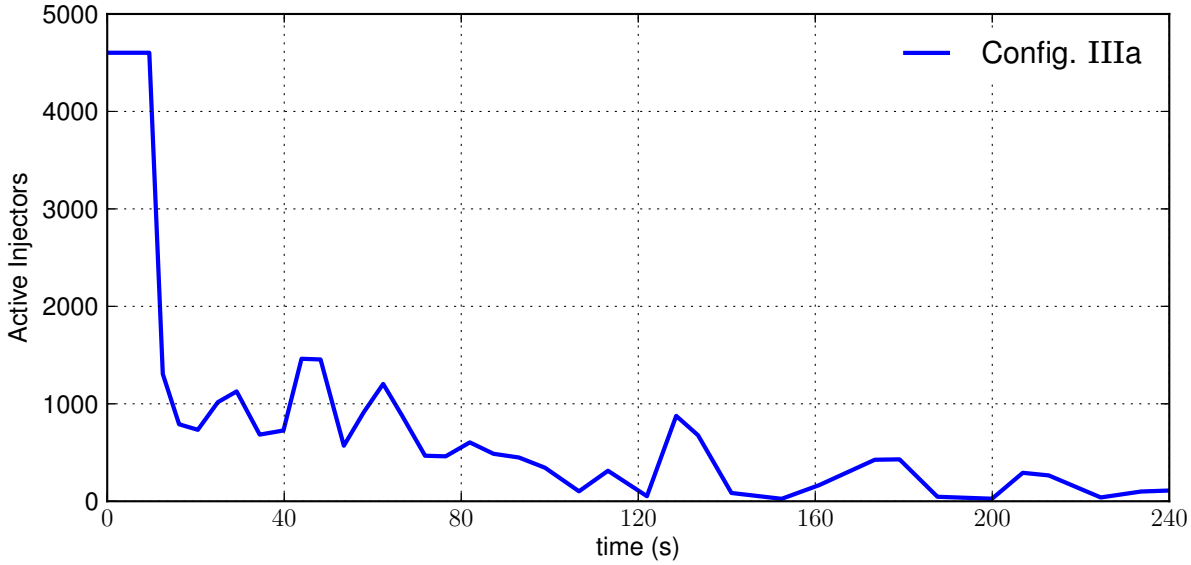


Figure 5.40: HQ: Number of active injectors

Figures 5.40 and 5.41 show respectively the number of active injectors and Satellite sub-domains during the simulation. In the short-term, all components remain active, thus the main speedup in this period comes from the parallelization of the DDM-based algorithm. In the long-term, the injectors start switching to latent followed by the Satellite sub-domains. However, it can be seen that even towards the end of the simulation, many Satellite sub-domains remain active. This is expected as the latter are part of the TN with large power exchanges with the Central sub-domain. These power flows, as well as response of a relatively large number of medium-size hydro generation plants connected to the Satellite sub-domains, cause the latter to frequently exceed $\epsilon_L = 0.1$ MVA and become or remain active. In Config. IIIc, many more Satellite sub-domains become latent thus achieving a speedup of

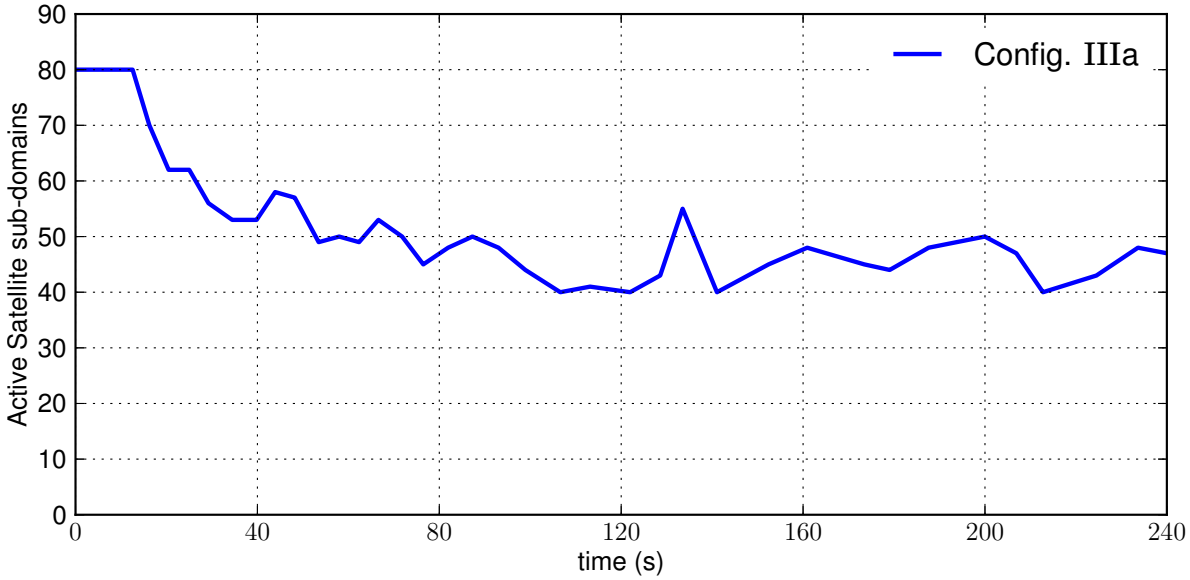


Figure 5.41: HQ: Number of active Satellite sub-domains

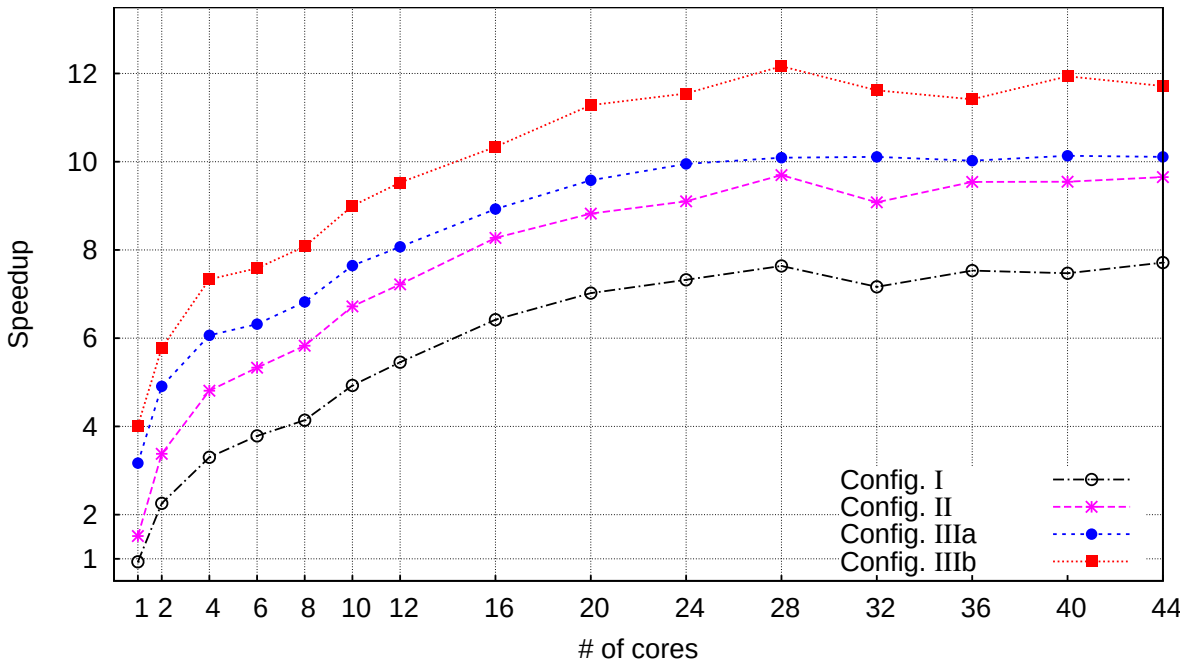


Figure 5.42: HQ: Speedup computed with Eq. 2.2

13 times; but, at the cost of higher inaccuracy.

From the parallel execution timings in Table 5.9, it can be seen that all configurations offer significant speedup when parallelized: up to 9.7 times without any inaccuracy and up to 11.7 times when latency is used and $\epsilon_L \leq 0.2$ MVA. A more detailed view is offered in Fig. 5.42, where the speedup is shown as a function of the number of cores used.

Figure 5.43 shows the scalability of both algorithms used for this test case. For Config. II, an extra scalability of 5-10% is observed in the two-level algorithm compared to the

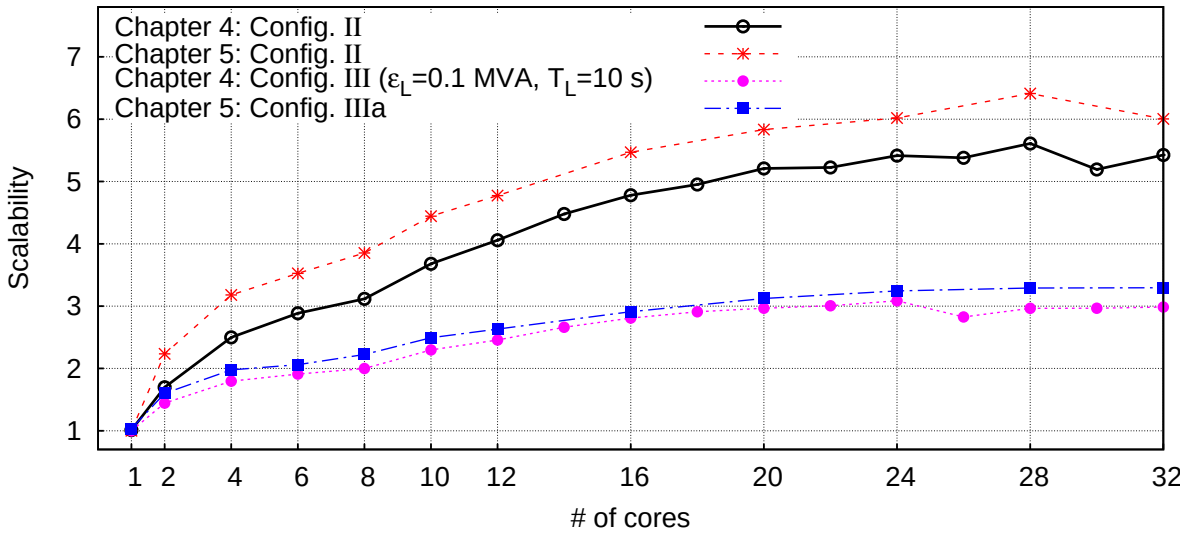


Figure 5.43: HQ: Effective scalability with the two decomposition algorithms

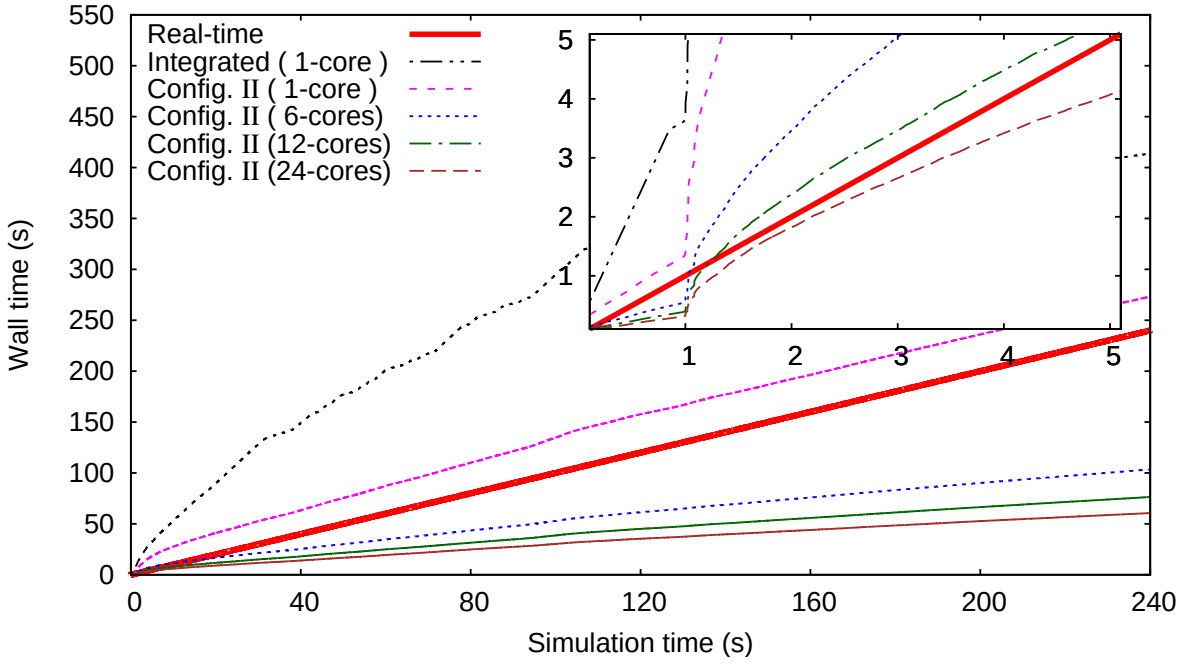


Figure 5.44: HQ: Real-time performance of algorithm

single-level DDM of Chapter 4. This gain comes from the higher parallelization percentages achieved by the two-level DDM *minus* the extra OHC associated with the two-level decomposition.

Finally, Fig. 5.44 shows the real-time performance of the algorithm with Config. II. Faster than real-time performance is achieved when executed on 24 or more cores, similarly to the algorithm of Chapter 4.

5.8.4 Discussion

The capabilities of the proposed two-level algorithm are discussed in this subsection. Its sequential and parallel performances are outlined, as well as some more results on the UMA Machines 2 and 3 of Section 2.7.

5.8.4.1 Sequential performance

Similarly to the algorithm of Chapter 4, the main sources of acceleration in sequential execution ($M = 1$) are the localization techniques. However, the proposed two-level DDM has some significant extra OHC compared to the integrated, as well as the single-level DDM.

Configuration I does not use any localization techniques and follows the same Jacobian matrix updates as the integrated. Except for the short-term test case of the first T&D system, the sequential execution of Config. I is slower than the integrated. As discussed in Section 4.9.4.1, this slowdown can be attributed to the extra OHC of the proposed DDM.

On the contrary, Configs. II and III take advantage of the localization techniques described in Section 5.5 and outperform the integrated, leading to fast sequential simulations.

5.8.4.2 Parallel performance

The proposed two-level algorithm offers significant speedup in parallel execution. Figures 5.17, 5.24, 5.31, and 5.42 show this speedup as a function of the number of cores used, reaching 22.2 times for full accurate simulations and much higher when latency technique is employed.

Even though the two-level algorithm has higher OHC than the previous algorithm (due to the more complex decomposition scheme), it has a larger percentage of work done in parallel (T_P). In addition to the treatment of injectors that was performed in parallel as in the single-level DDM, this algorithm also treats in parallel the Satellite sub-domain networks. Thus, the sequential bottleneck coming from of the Schur-complement approach of treating the interface variables is limited to solving the global reduced system (5.15). The size of the latter sparse linear system is double the number of buses of the Central sub-domain network.

This large percentage of parallel work leads to increased parallelization efficiency. That is, the proposed algorithm scales over a larger number of computational cores than the previous one and allows to better exploit the available computational resources.

5.8.4.3 Performance with UMA standard office laptops

For the previous simulations, Machine 1 was used to “scan” a varying number of cores and show the performance and scalability of the algorithm. However, the proposed two-level DDM can provide significant speedup even on smaller UMA machines. For this reason, the scenario of Section 5.8.1 was executed on Machines 2 and 3 to show the performance of the algorithm. The dynamic response is not presented as it is identical to the one shown previously.

Table 5.10: Nordic variant 1, Scenario 2a: Execution times of UMA machines

	Sequential execution time ($M = 1$) (seconds/speedup)		Fastest parallel execution time (seconds/speedup)	
Machine (see Section 2.7)	2	3	2	3
Integrated (T_1^*)	532 / -	464 / -	- / -	- / -
Config. I	677 / 0.8	580 / 0.8	423 / 1.3	185 / 2.5
Config. II	449 / 1.2	371 / 1.3	264 / 2.0	128 / 3.6
Config. IIIa	384 / 1.4	299 / 1.6	269 / 2.0	104 / 4.5
Config. IIIb	326 / 1.6	243 / 1.9	209 / 2.6	88 / 5.3
Config. IIIc	192 / 2.8	149 / 3.1	129 / 4.1	59 / 7.9

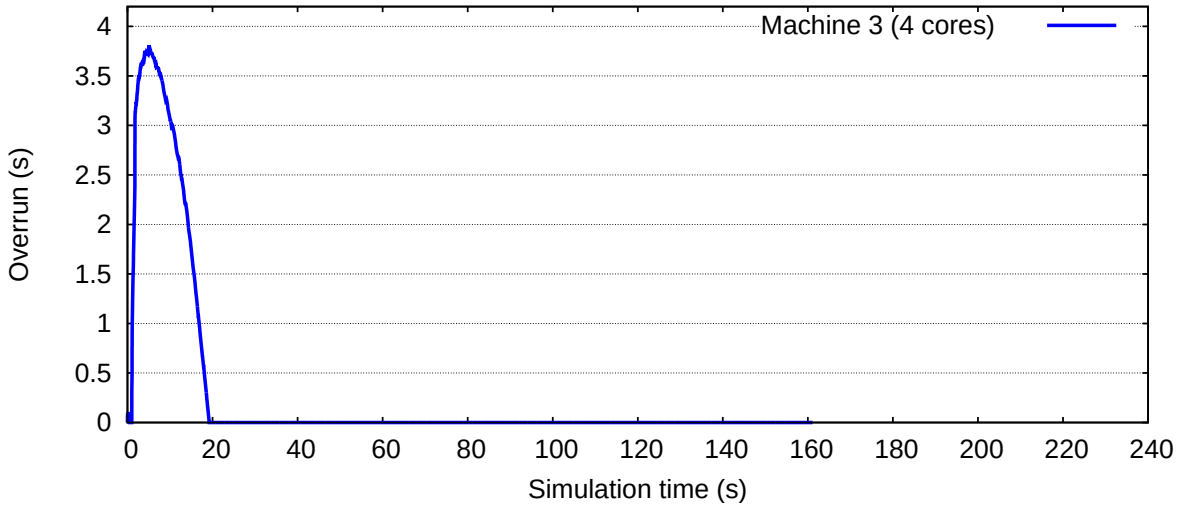


Figure 5.45: Nordic variant 1, Scenario 2a: Overrun of the algorithm on Machine 3 with Config. II

Table 5.10 shows the execution times and speedup of the simulation on the aforementioned laptops. As with the results of Section 5.8.1, in the sequential execution with Config. I, the proposed two-level DDM shows some slow-down due to its OHC. However, on Machine 2 (resp. 3) the sequential execution offers a speedup of 1.2 (resp. 1.3) for Config. II and 1.6 (resp. 1.9) for Config. IIIb.

In parallel execution, a speedup of 2.0 (resp. 3.6) is achieved with Config. II using the two (resp. four) cores of the laptop. Configuration IIIb shows a speedup of 2.6 (resp. 5.3), simulating this 14500-bus (modeled with 143000 DAEs) system in 209 s (resp. 88 s). Figure 5.45 shows the overrun of the simulation on Machine 3 in parallel execution. It can be seen that there is an overrun of 3.5 s, thus, real-time simulations are possible for this system.

Overall, the proposed two-level DDM can provide significant speedup on normal laptop computers, thus allowing to perform fast and accurate power system dynamic studies without the need of expensive equipment.

5.9 Summary

The DDM-based algorithm presented in this chapter relies on a two-level decomposition. The first decomposition partitions the network to reveal a star-shaped layout. Then, the injectors are isolated from each sub-network, similarly to the single-level algorithm of Chapter 4. The sub-domains formulated on both levels are treated independently and in parallel while their interface variables are updated hierarchically using a Schur-complement approach. Furthermore, three localization techniques were presented to accelerate the simulation both in sequential and parallel execution.

This algorithm allows to accelerate the simulation of T&D systems, by exploiting their particular structure and employing an hierarchical, two-level, parallel DDM. Moreover, the use of Schur-complement approach to treat the interface variables at each decomposition level, allows to retain high convergence rate of the algorithm. Finally, the sequential bottleneck of the Schur-complement-based approach can be significantly decreased with the proper selection of the Central sub-domain, allowing to achieve high scalability.

CHAPTER 6

General conclusion

6.1 Summary of work and main contributions

Power system dynamic simulations have many applications, such as dynamic security assessment, simulation-driven design, planning, operation, etc. Such simulations involve the solution of an initial value problem, described by a system of hybrid, nonlinear, stiff Differential-Algebraic Equations (DAEs), over a specific time window. The large size and the hybrid nature of the system (including continuous and discrete states), as well as the stiffness of the equations, make the problem challenging to tackle and time consuming.

This work brings together elements from numerical analysis, Domain Decomposition Methods (DDMs), and parallel computing, to accelerate the dynamic simulation of power systems. Two algorithms have been proposed belonging to the family of DDMs.

The first algorithm targets large interconnected transmission systems, without considering any particular network topology. The decomposition applied, partitions the system into the network and the injectors connected to it. This decomposition reveals a star-shaped sub-domain layout and leads to the separation of the DAEs describing the system. Then, the nonlinear DAE sub-systems are solved independently by algebraizing and using a Newton method with infrequent matrix update and factorization. The interface variables shared between sub-domains are updated using a Schur-complement approach at each iteration.

The second algorithm targets systems with a particular radial network topology. Mainly, it was designed to accelerate the simulation of detailed combined Transmission and Distribution (T&D) systems, where each of the latter is connected to the transmission network at one bus. However, it was shown that some sub-transmission systems with partially radial structure can also benefit from this algorithm. This DDM applies a two-level system decomposition. First, the system is decomposed on the boundary between the transmission and distribution networks. This leads to the creation of several sub-domains, each defined by its own network and injectors. Then, a second decomposition is applied within each sub-domain, splitting the network from the injectors, in a similar way to the single-level algorithm. Finally, the sub-domain DAE systems are solved hierarchically using a Newton method with

infrequent matrix update and factorization for each sub-system. The interface variables are updated using a Schur-complement approach, at each decomposition level, at each iteration.

In Sections 4.7 and 5.6, it was shown that the proposed DDMs are mathematically equivalent to solving the nonlinear DAEs with a unique (integrated) *quasi-Newton method*. This feature is due to the Schur-complement approach used to update the interface variables at each sub-system iteration. This observation allows to use the extensive theory behind quasi-Newton schemes to assess the proposed DDMs convergence.

Both proposed DDMs have been accelerated computationally and numerically.

First, the independent calculations of the sub-systems have been parallelized providing computational acceleration. This parallelization potential is inherent to DDMs and the main reason for using such methods. DDMs using a Schur-complement approach to update the interface variables, such as the ones proposed in this work, suffer from an unavoidable sequentiality that can hinder their parallel performance. However, it was shown that selecting a star-shaped decomposition, exploiting the sparsity of the central sub-domain, and using an infrequent update of the reduced matrix, minimizes the computational cost of this sequential task and allows for higher scalability. In this work, the OpenMP shared-memory parallel computing API has been used to implement both algorithms, thus allowing to exploit modern multi-core computers. A large multi-core computer, setup and maintained for the purpose of this work, was used to asses the performance of the algorithms.

Second, three numerical acceleration techniques have been used to exploit the locality of the decomposed sub-systems and avoid unnecessary computations. The first technique, considers skipping the solution of sub-systems that have already converged to the required accuracy within one time-step solution. The second technique, suggests the asynchronous update of sub-system matrices and of the reduced systems. The last technique, replaces the dynamic models of sub-systems with low dynamic activity, with linear, sensitivity-based, equivalent ones computed at the moment of switching. On the contrary, when high dynamic activity is detected, the original dynamic models are reinstated. In this work, these three techniques have been extended and enhanced to consider the two-level decomposition algorithm. Moreover, a new metric, stemming from real-time digital signal processing, has been proposed to quantify the level of sub-system dynamic activity.

Furthermore, it was shown that the proposed DDMs using these acceleration techniques, are mathematically equivalent to solving the nonlinear DAEs with a unique (integrated) *Inexact Newton* method. The first two acceleration techniques do not disturb the accuracy of the simulation response. However, the third technique, *latency*, allows to achieve high simulation performance while sacrificing some accuracy. In this case, the compromise between speed and accuracy can be easily tuned using only a couple of parameters (latency tolerance and time window observation).

The performance and accuracy of the proposed DDMs have been tested on the power system models presented in Section 1.3. It was shown that the proposed algorithms can achieve high speedup and scalability, both on scientific computing machines, as well as on

normal laptop computers. Next, the real-time capabilities of the DDMs were examined, and it was shown that in all systems, applications with "soft" real-time requirements (allowing some limited overrun) can be envisioned. In addition, for some of the systems studied (Nordic and HQ), applications demanding "hard" real-time (no overrun) are also possible.

Finally, it was shown that in long-term scenarios, combining the localization and parallel computing techniques, provides the best simulation performance. On the one hand, when the system exhibits high dynamic activity (usually during the short-term dynamics), the parallel computing techniques offer higher speedup as more frequent matrix updates and system solutions are necessary, which are performed in the parallel sections of the algorithms. On the other hand, the localization techniques perform better when the system evolves smoothly, without *much* dynamic activity (usually during the long-term dynamics).

The algorithms proposed in this work have been implemented in the dynamic simulation software RAMSES. The latter is currently used by researchers at various universities and research institutes as their main dynamic simulation and development platform, e.g.:

- Design and validation of Active Distribution Network (ADN) control schemes (University of Liège – Belgium, École Centrale de Lille – France, University of Costa Rica);
- Robustness and defense plans in mixed AC-DC systems (University of Liège – Belgium, a research supported by the R&D department of RTE[1]);
- Performing power hardware-in-the-loop experiments to study the integration of electric vehicles to ADNs (ENSAM[2] – France);
- Development of mixed-signal computer for power system applications (EPFL[3] - Switzerland);
- Development of a high-performance dynamic security assessment platform with the target of 100,000 dynamic simulations per day in operational planning by 2016 (Hydro-Québec, TransÉnergie division and IREQ[4] – Canada); and,
- Voltage stability analysis and preliminary reactive power reserve planning of the future German grid (just launched collaboration with Amprion[5]).

To conclude, the current trend for power system dynamic simulations is to demand more detailed, and thus bigger and more complex, injector models. In addition, the most noticeable developments foreseen in power systems involve Distribution Networks (DNs), which are expected to host a big percentage of the renewable energy sources. Thus, DN equivalencing will become more and more difficult, and the need for detailed representation more acute.

[1] The French TSO
[2] École Nationale Supérieure d'Arts et Métiers
[3] École Polytechnique Fédérale de Lausanne
[4] Institut de recherche en électricité du Québec
[5] The largest TSO in Germany

The proposed DDMs decompose power system models in such way that the future increase in computational demand, due to the previous observations, will increase the percentage of work in the parallel sections of the algorithms. In that way, Gustafson's law is validated. That is, using the same algorithms and computational equipment, it is expected that with future demands in power system modeling, the DDMs will provide even higher scalability and speedup.

6.2 Directions for future work

The presented algorithms may be further improved or find new application along the following directions:

- One of the main requirements of the proposed two-level DDM is that there is only one point of connection between the Central and Satellite sub-domains. The benefit of having Satellite sub-domains with only one point of connection is that during the elimination procedure for the formulation of the global reduced system, the original sparsity pattern of the Central sub-domain network is preserved. However, in some power systems, there exist sub-transmission networks that are connected to the bulk transmission system at several buses. A modified two-level algorithm can be designed to exploit these sub-transmission networks as Satellite sub-domains to accelerate the simulation procedure. In this case, their Schur-complement terms will introduce some fill-ins to the sparsity of the global reduced system, similarly to the twoports in the reduced system of the single-level algorithm.

- The latency-localization technique used by both the proposed algorithms, provides significant acceleration at the cost of introducing some inaccuracy to the simulated response. One of the sources of inaccuracy is the linear sensitivity-based model used when latent. The latter represents only the interface of the injector or the Satellite sub-domain to the transmission network. That is, the linear model involves and updates only the two interface states, while all the other states of the model are frozen at the moment of becoming latent. Consequently, when the model switches from latent to active again, there is an inconsistency between the interface and the remaining states, which creates a discrete jump in the equations and can disturb the quasi-Newton method.

  To avoid the latter problem, a new linear sensitivity-based model can be used that updates all the states of the latent component. This way, when switching from latent to active, the states of the model will be all updated and the equations consistent. Nevertheless, this entails the extra cost of updating larger linear models.

- The proposed DDMs can be also used in other power system applications involving linear systems derived from the same power system models. One such computationally demanding task, is the eigenvalue analysis of large-scale power systems. In [RM09], it

was proposed to exploit the sparsity of the power system Jacobian matrix used for the calculation of the eigenvalues, in order to accelerate the calculation of the rightmost eigenvalues. The main computational effort of methods like the Implicitly Restarted Arnoldi with Shift-and-Invert [RM09], is the (repetitive) solution of linear systems of the form:

$$(\boldsymbol{J}_S - \sigma \boldsymbol{I})\boldsymbol{x} = \boldsymbol{b} \tag{6.1}$$

where $\boldsymbol{J}_S$ is the Jacobian state-space matrix derived after the elimination of the algebraic states, $\boldsymbol{I}$ is the unit matrix, and $\sigma \epsilon \mathbb{C}$ is the shift. It was shown [RM09] that for better performance, the latter equation can be rewritten as:

$$(\boldsymbol{J} - \sigma \boldsymbol{\Gamma} \boldsymbol{I})\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}} \tag{6.2}$$

where $\boldsymbol{J}$ is the Jacobian descriptor matrix, $\boldsymbol{\Gamma}$ is a diagonal matrix as defined in (1.2), and:

$$\tilde{x}_i = \begin{cases} x_i & \text{if } i \text{ is differential} \\ ignored & \text{if } i \text{ is algebraic} \end{cases} \qquad \tilde{b}_i = \begin{cases} b_i & \text{if } i \text{ is differential} \\ 0 & \text{if } i \text{ is algebraic} \end{cases} \tag{6.3}$$

The benefit is that $\boldsymbol{J}$ (contrary to $\boldsymbol{J}_S$) is a sparse matrix with a structure similar to that of $\boldsymbol{J}$ in Eqs. 4.20 or 5.19, thus fast sparse solvers can be used. However, based on the observations made in Sections 4.7 and 5.6, an equivalent solution of a system with the form (6.2) can be performed by the single-level or two-level algorithms and thus the use of parallel computing techniques could be envisaged to accelerate its solution.

Appendices

APPENDIX **A**

Analysis of Newton-type schemes

A.1 Review

In numerical analysis, Newton's method (also known as the Newton–Raphson method), named after Isaac Newton and Joseph Raphson, is a root-finding algorithm that uses the first few terms of the Taylor series of a function $f(y)$ in the vicinity of a suspected root (y_0).

The Taylor series of $f(y)$ about the point $y_1 = y_0 + \epsilon$ is given by:

$$f(y_1) = f(y_0) + \dot{f}(y_0)(y_1 - y_0) + \frac{1}{2}\ddot{f}(y_0)(y_1 - y_0)^2 + ... \tag{A.1}$$

Ignoring the higher-order terms, we obtain:

$$f(y_1) \approx f(y_0) + \dot{f}(y_0)(y_1 - y_0) \tag{A.2}$$

This expression above can be used to estimate the correction needed to land closer to the root starting from an initial guess y_0. Setting $f(y_1) = 0$ and solving for $y_1 - y_0$ gives:

$$y_1 = y_0 - \frac{f(y_0)}{\dot{f}(y_0)} \tag{A.3}$$

which is the first-order adjustment to the root's position. The process can be repeated until it converges to a fixed point (which is precisely a root) using:

$$y_{n+1} = y_n - \frac{f(y_n)}{\dot{f}(y_n)} \tag{A.4}$$

for $n = 0, 1, 2, ...$

This method can be generalized to a system of m algebraic equations with m unknowns $\boldsymbol{F}(\boldsymbol{y}) = \boldsymbol{0}$. Everything remains the same, except that the first derivative of $\boldsymbol{F}$ is replaced by the $m \times m$ Jacobian matrix $\boldsymbol{J} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}$. Thus, the iterations are:

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n - \left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}(\boldsymbol{y}_n)\right)^{-1} \boldsymbol{F}(\boldsymbol{y}_n) \tag{A.5}$$

for $n = 0, 1, 2, ...$

In real applications, the inverse Jacobian matrix is never computed. Rather than computing directly $\boldsymbol{y}_{n+1}$, the correction $\Delta\boldsymbol{y}_n = \boldsymbol{y}_{n+1} - \boldsymbol{y}_n$ is calculated by solving the linear system:

$$\underbrace{\left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}(\boldsymbol{y}_n)\right)}_{\boldsymbol{J}_n} \Delta\boldsymbol{y}_n = -\boldsymbol{F}(\boldsymbol{y}_n) \tag{A.6}$$

and then updating the variables:

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \Delta\boldsymbol{y}_n \tag{A.7}$$

When the Jacobian matrix is sparse [TW67], such in the case of power system problems, the use of sparse linear solvers makes the solution of (A.6) very efficient.

The most computationally expensive part of the solution of (A.6) is the calculation of the Jacobian matrix and its factorization. Thus, to increase the method's performance, an infrequent update and factorization can be employed. That is, the Jacobian matrix is kept the same for several consecutive solutions and it is only updated after a predefined number of iterations or when the method fails to converge. These methods are called quasi-Newton or perturbed Newton in mathematics [Kel95] and are referred to as Very DisHonest Newton (VDHN) in power systems [Mil10].

Both the exact and the quasi-Newton methods have been extensively studied and their convergence criteria and capabilities have been presented in several publications [Bro70, BDM73, DM74, GT82, DW84, Kel95].

A.2 Inexact Newton schemes

In some cases, there is inaccuracy in the system formulated in (A.6), leading to an *inexact* solution. This inaccuracy can be unintentional, for example due to the linear solver or machine precision, or intentional with the aim of improving the efficiency of the overall, combined process [DES82]. To depict this, Eqs. A.6 and A.7 can be rewritten as:

$$\boldsymbol{J}_n \delta\boldsymbol{y}_n = -\boldsymbol{F}(\boldsymbol{y}_n) - \boldsymbol{r}_n \tag{A.8}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \delta\boldsymbol{y}_n$$

where $\boldsymbol{r}_n$ is the inaccuracy. These schemes are referred to as Inexact Newton (IN) [Cat04].

Combining (A.6) and (A.8) leads to:

$$\boldsymbol{J}_n \left(\delta\boldsymbol{y}_n - \Delta\boldsymbol{y}_n\right) = -\boldsymbol{r}_n \tag{A.9}$$

Thus the difference between the IN scheme and the exact Newton method can be measured by either one of the following relative errors [DES82]:

$$\frac{\|\delta\boldsymbol{y}_n - \Delta\boldsymbol{y}_n\|}{\|\delta\boldsymbol{y}_n\|} \qquad \text{or} \qquad \frac{\|\boldsymbol{r}_n\|}{\|\boldsymbol{F}(\boldsymbol{y}_n)\|} \tag{A.10}$$

Of course, it depends on which one of these error measures is available or can be estimated during the course of the process.

The required criteria for the IN scheme to converge to the same value as the exact Newton have been shown in [DES82]. Briefly, if the normal convergence requirements for Newton methods [DES82] are satisfied and in addition:

$$\frac{\|\boldsymbol{r}_n\|}{\|\boldsymbol{F}(\boldsymbol{y}_n)\|} < \eta < 1 \qquad \forall n \geq 0 \tag{A.11}$$

then, the IN scheme converges to the same value as the exact Newton.

It is noteworthy that the quasi-Newton methods described in the previous section can be also analyzed as IN methods. Let us consider the following quasi-Newton (or VDHN) method:

$$\tilde{\boldsymbol{J}}_n \delta \boldsymbol{y}_n = -\boldsymbol{F}(\boldsymbol{y}_n) \tag{A.12}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \delta \boldsymbol{y}_n$$

where $\tilde{\boldsymbol{J}}_n = \boldsymbol{J}_n + \Delta \boldsymbol{J}_n$ is the outdated Jacobian and $\Delta \boldsymbol{J}_n$ is the error from the correct (updated) Jacobian. Equation A.12 can then be rewritten as:

$$\left(\boldsymbol{J}_n + \Delta \boldsymbol{J}_n\right) \delta \boldsymbol{y}_n = -\boldsymbol{F}(\boldsymbol{y}_n)$$

$$\implies \quad \boldsymbol{J}_n \delta \boldsymbol{y}_n = -\boldsymbol{F}(\boldsymbol{y}_n) - \underbrace{\Delta \boldsymbol{J}_n \delta \boldsymbol{y}_n}_{\boldsymbol{r}_n} \tag{A.13}$$

which can then be treated and analyzed as an IN scheme.

APPENDIX **B**

Test-System diagrams

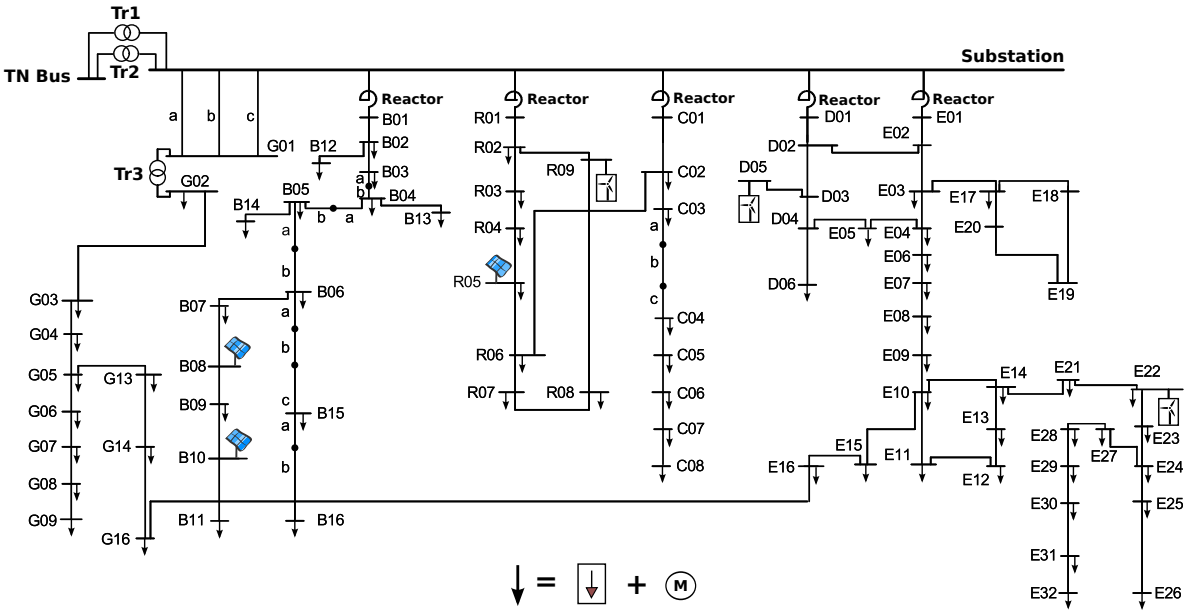


Figure B.1: Lelystad distribution system

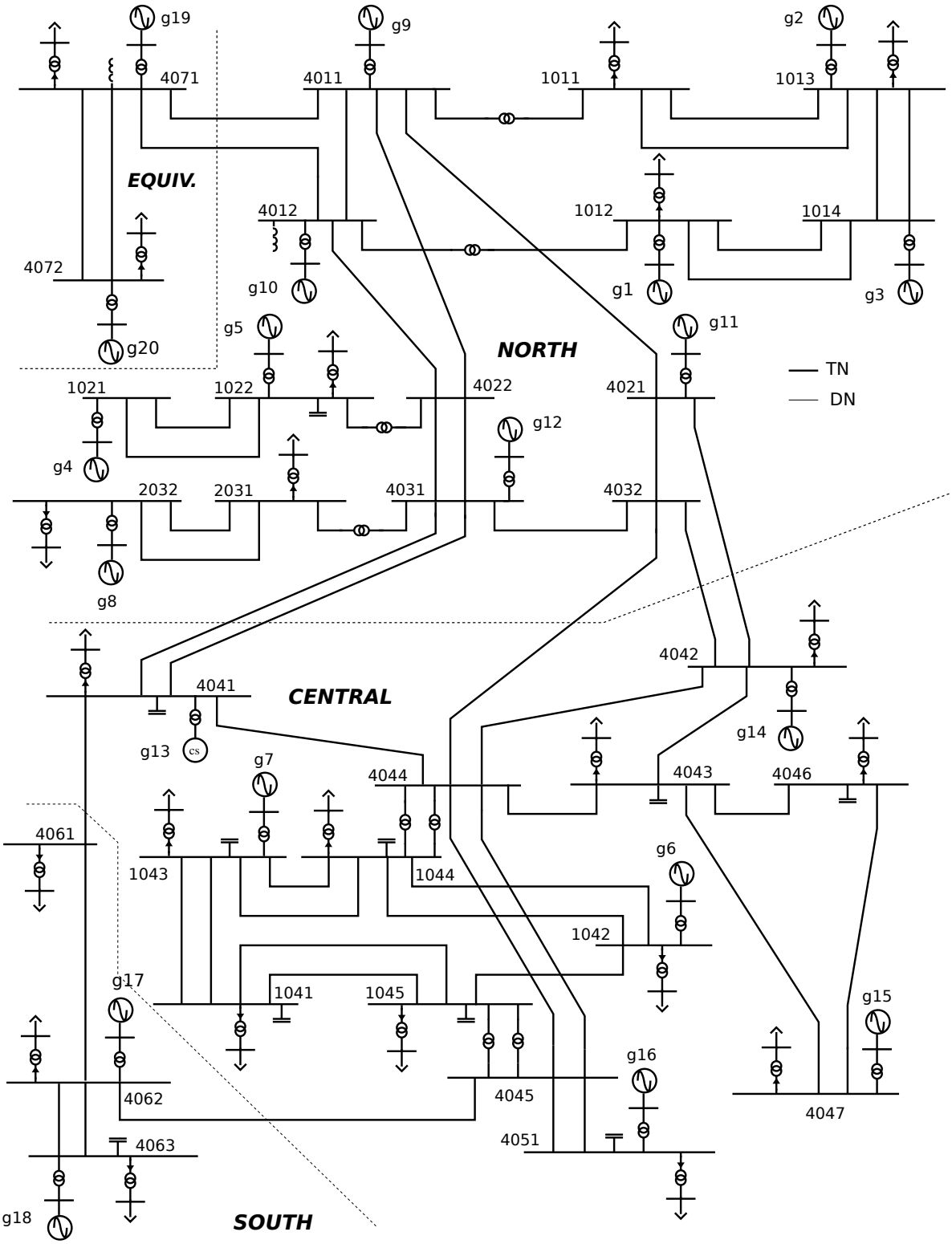


Figure B.2: Nordic32 system

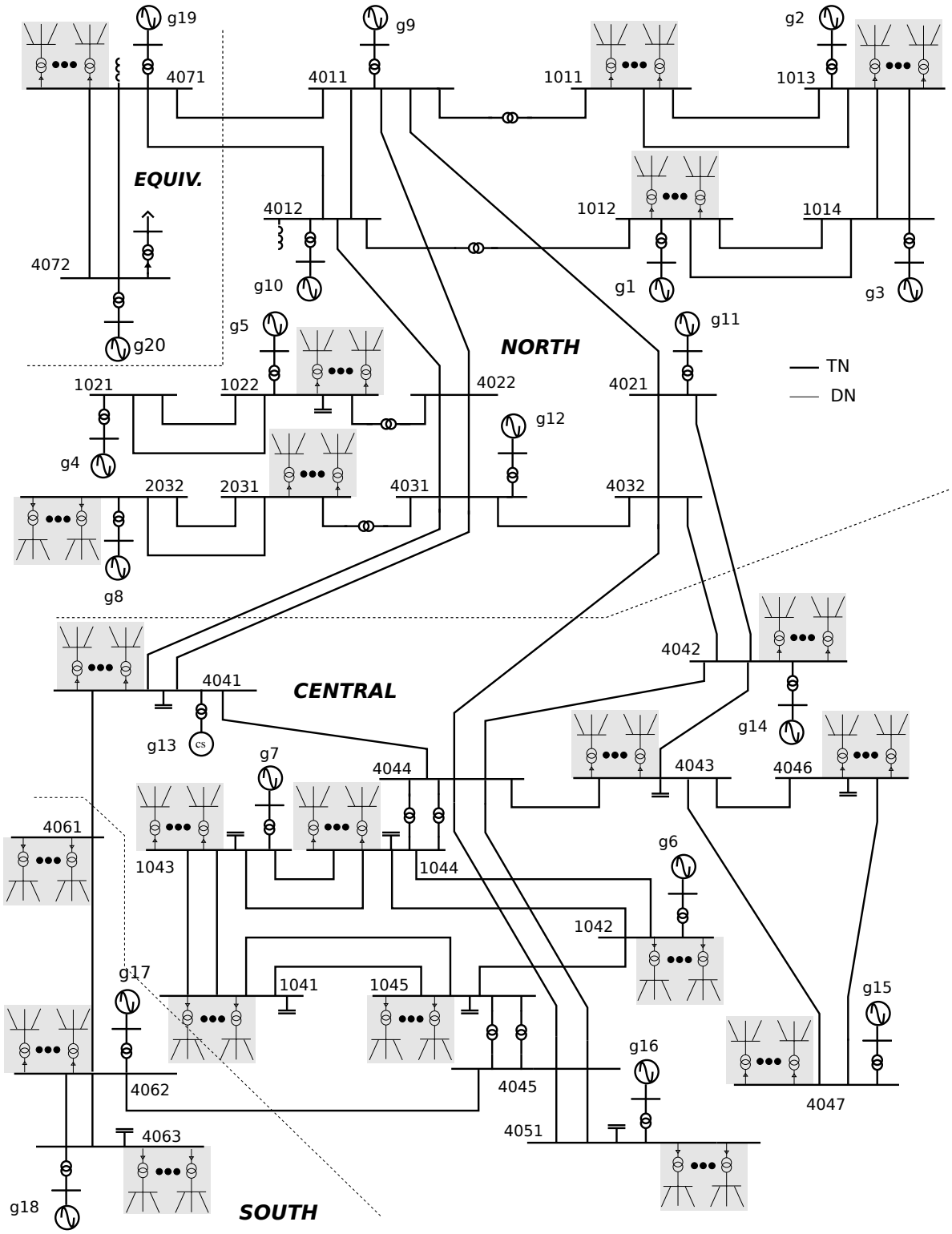


Figure B.3: Nordic variant 1: Nordic32 system expanded with 146 Lelystad distribution systems

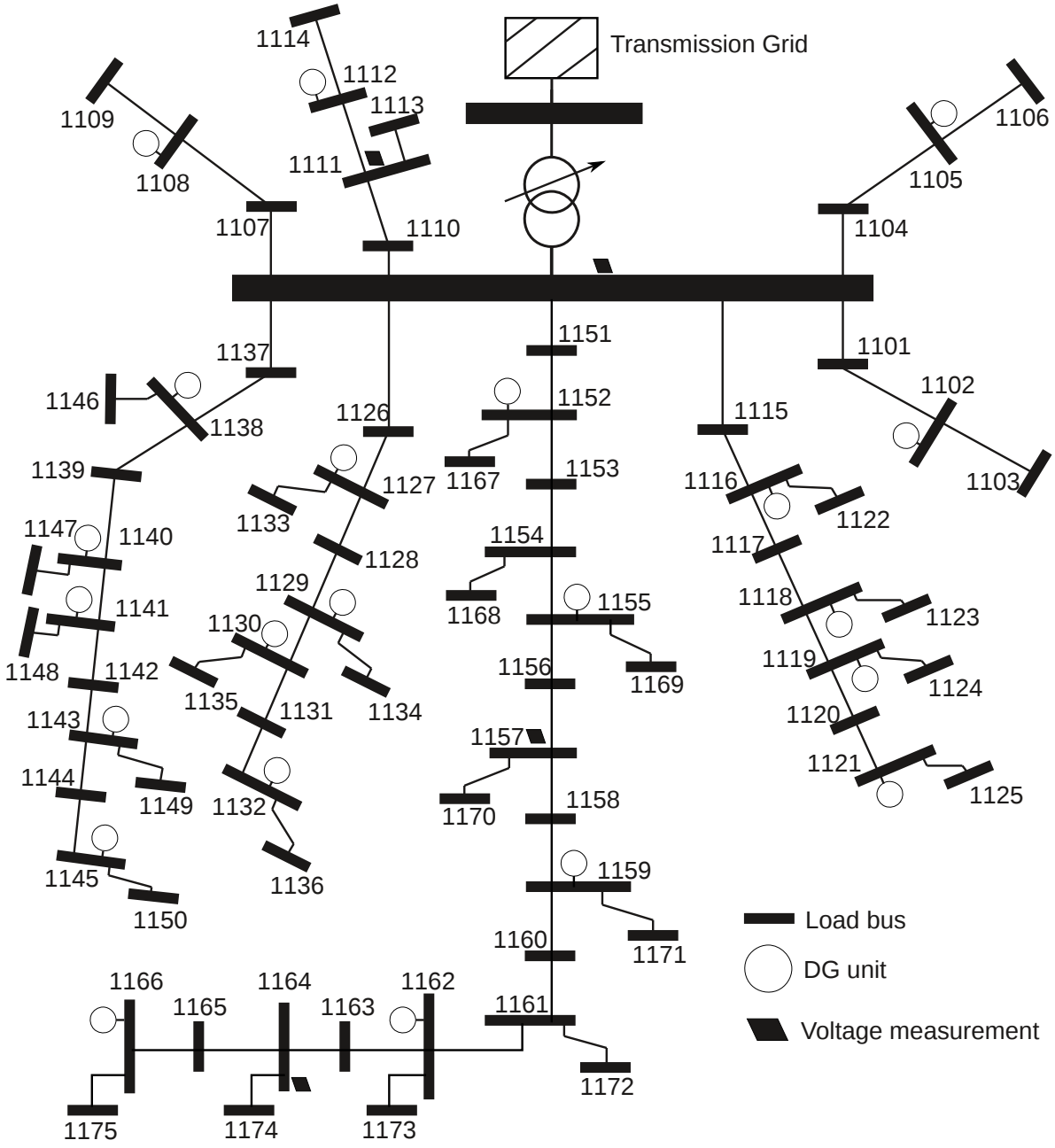


Figure B.4: 75-bus distribution system

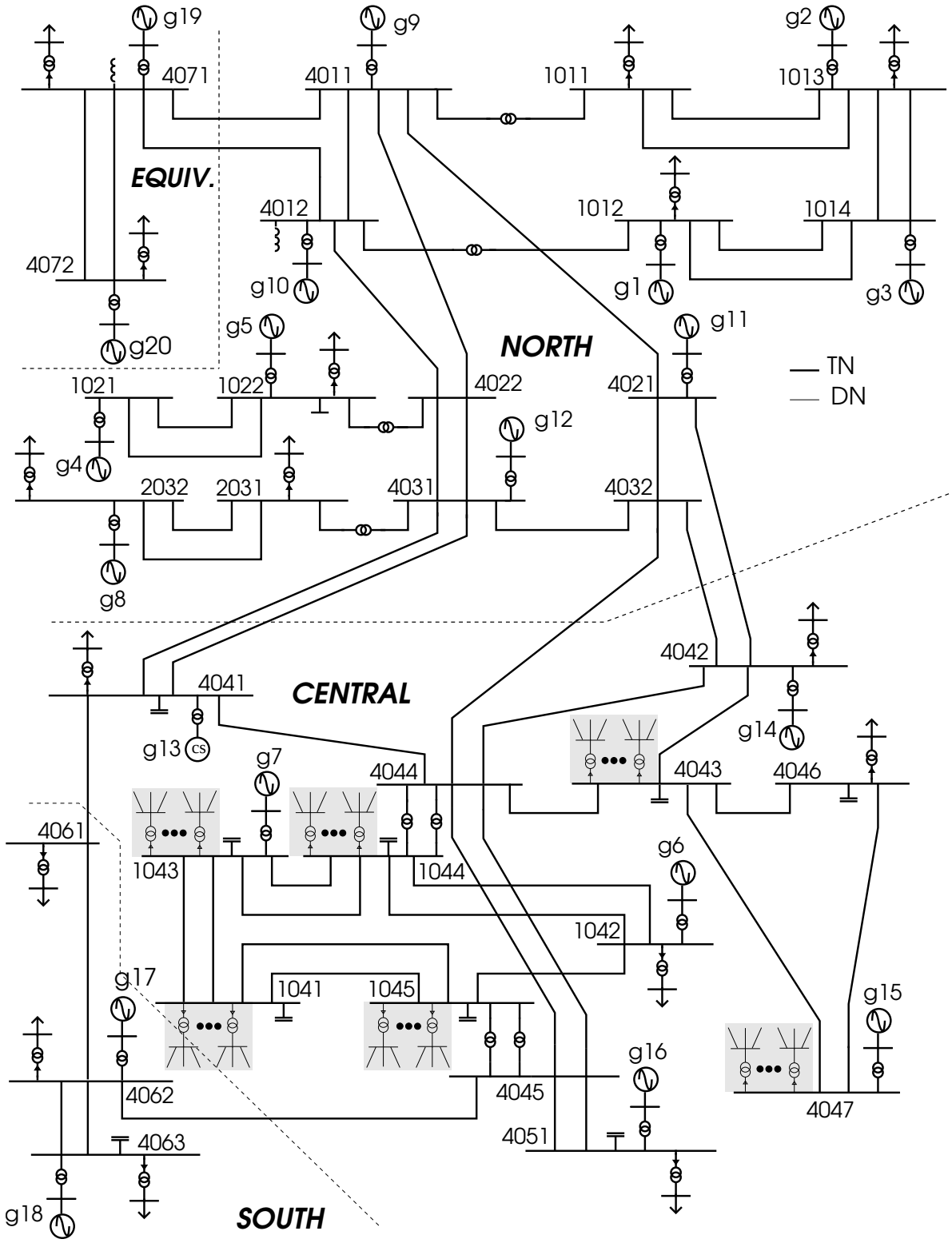


Figure B.5: Nordic variant 2: Nordic32 system expanded with 40 75-bus distribution systems

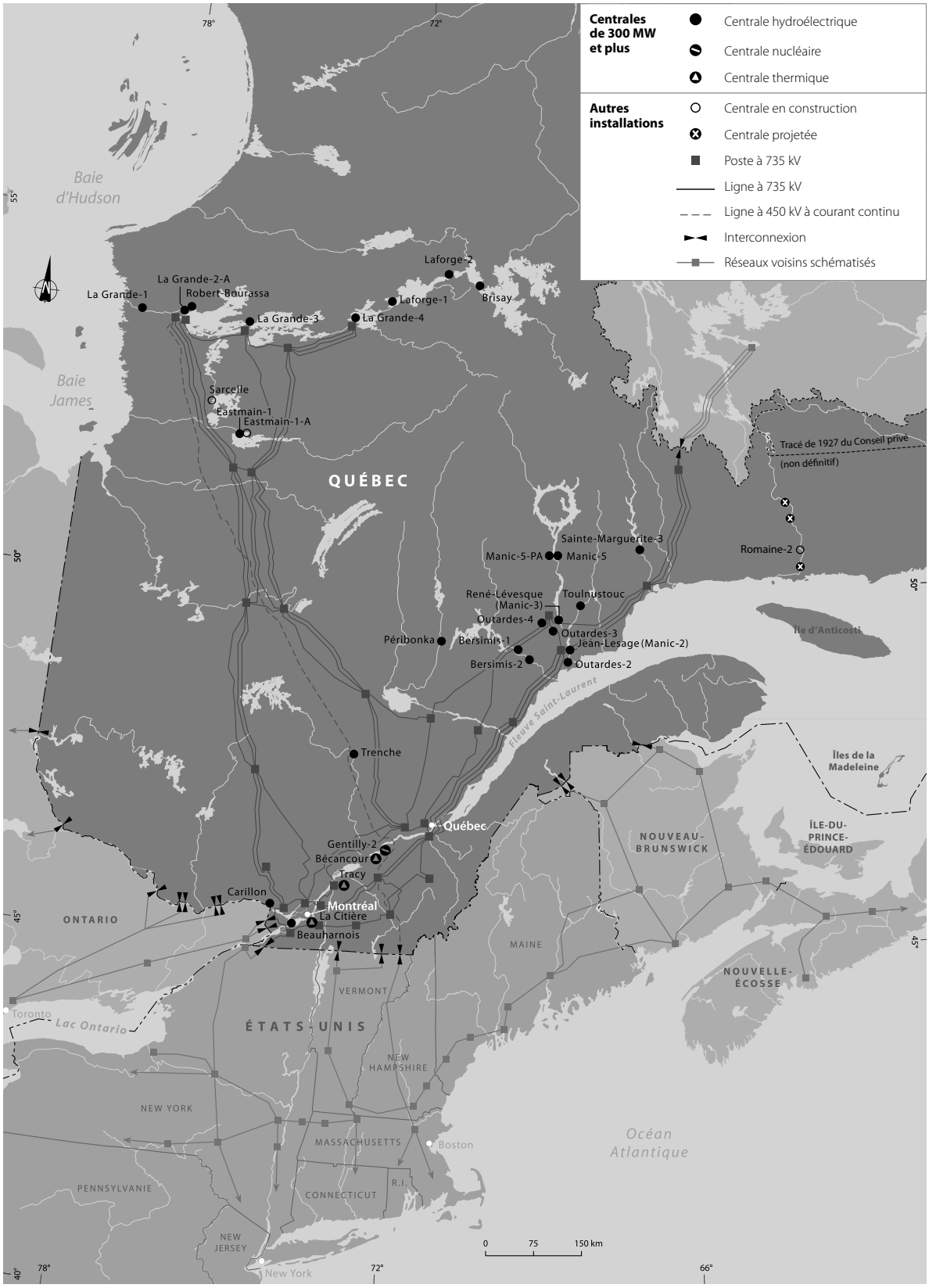


Figure B.6: Map of the Hydro-Québec system (as of 2010)

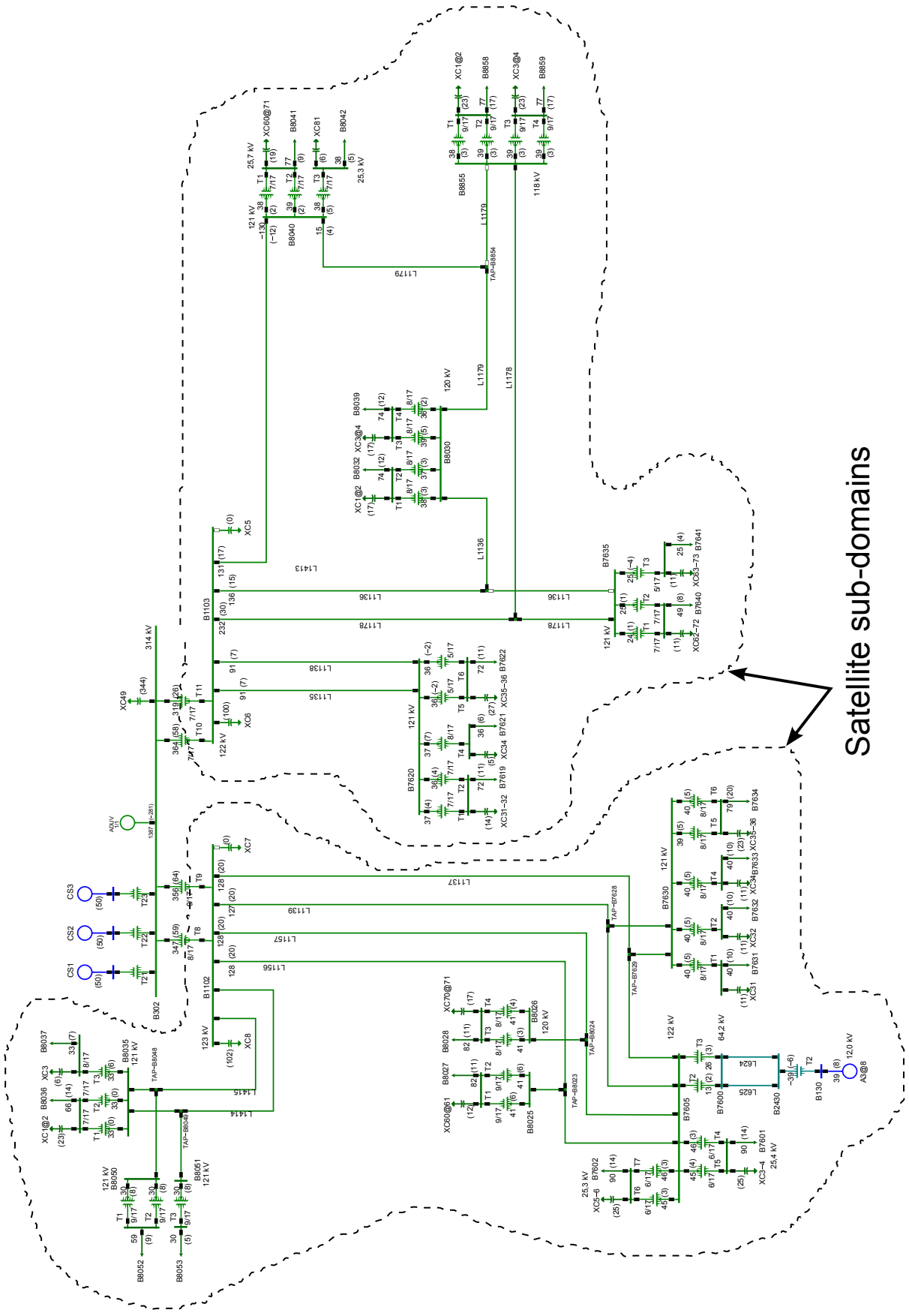


Figure B.7: Example of Hydro-Québec radial sub-systems used as Satellite sub-domains

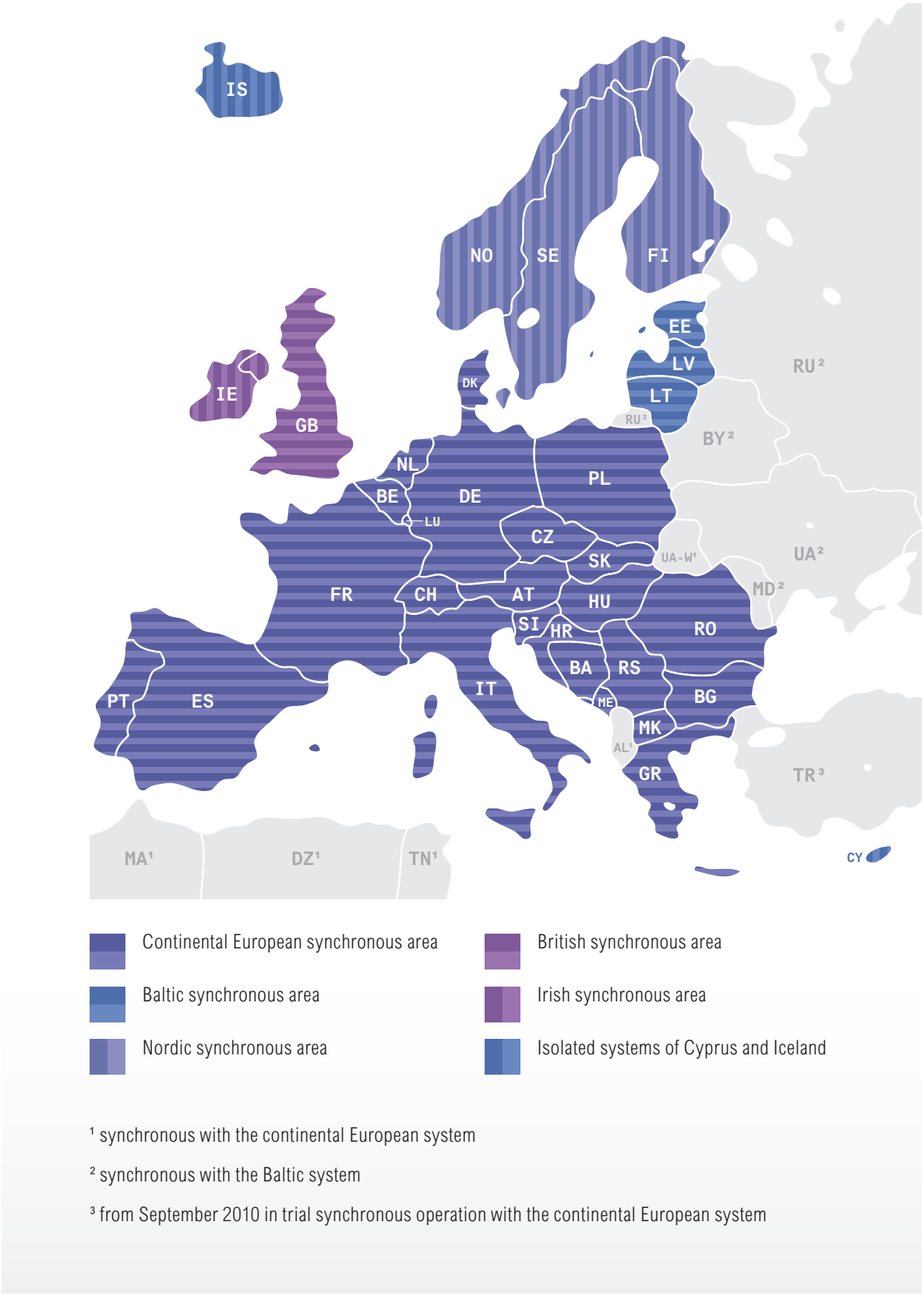


Figure B.8: European map with indication of the interconnected ENTSO-E system members (2011); continental European synchronous area is represented in the PEGASE system

APPENDIX C

RAMSES

C.1 Introduction

Power systems are equipped with more and more controls reacting to disturbances, with either beneficial or detrimental effects. Thus, static security analysis is no longer sufficient and dynamic responses need to be simulated. Moreover, larger and larger models are considered deriving form the simulation of large interconnections or the incorporation of sub-transmission and distribution levels. In addition, longer simulation times need to be considered in order to check the response of system up to its return to steady state (long-term dynamics can take up to several minutes after initiating event). Finally, there is a demand for faster than real-time simulators with look-ahead capabilities (hardware or controller-in-the-loop tests, closed-loop remedial actions in control centers, etc.).

To cover this need for faster power system dynamic simulations, a simulator called RAMSES (stands for "RApid Multithreaded Simulator of Electric power Systems") is developed at the University of Liège since 2010. The main developers are Petros Aristidou, Davide Fabozzi, and Thierry Van Cutsem. There are three contributing components to the development of this software (see Fig. C.1), all of which are tightly coupled to the DDM-based simulation algorithms used. The first, is how the power system is modeled. The second relates to the acceleration techniques used to provide fast and accurate simulation responses. The final component is the software implementation, that is, how the simulator is implemented and how the user interfaces with it.

In the remaining of this appendix, some of these aspects will be summarized.

C.2 Power system modeling

Based on the power system decomposition presented in Chapter 4, a power system model in RAMSES consists of the injectors, twoports, AC and DC networks, as seen in Fig. C.2. Each injector (or twoport) is interfaced to the network through the (rectangular components of) bus voltage and injected current, and is modeled with its own hybrid Differential-Algebraic

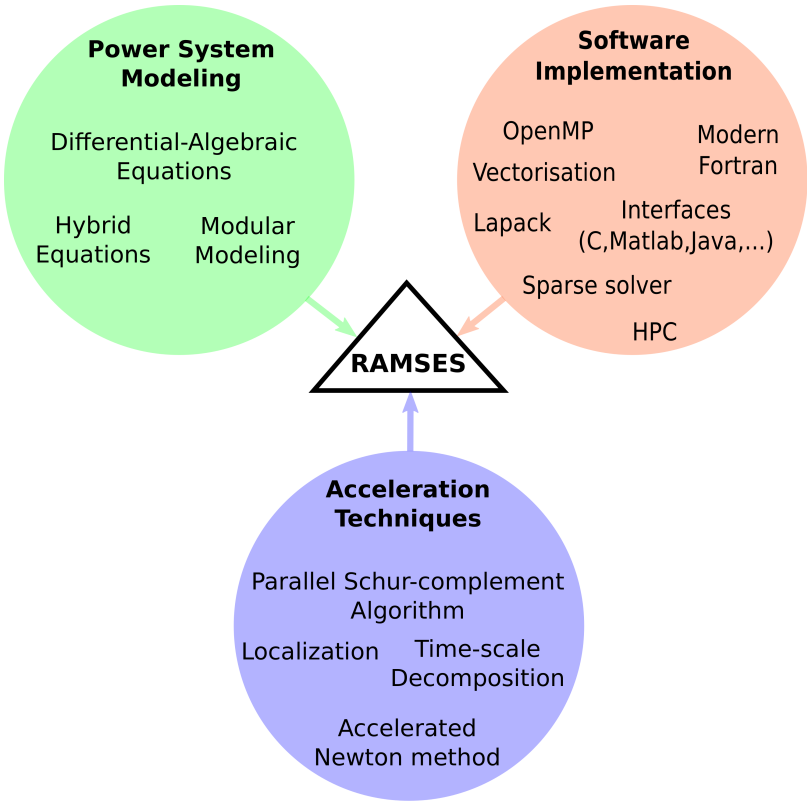


Figure C.1: Consisting elements of RAMSES

Equations (DAEs). Allowing for algebraic equations in the models yields higher flexibility. Furthermore, the equations can change between differential and algebraic during the simulation, due to discrete changes in the model. For an injector, the interfacing is shown in Fig. C.3.

When the decomposition presented in Chapter 5 is used, the AC network is also split into the Central and the Satellite sub-domains.

Both decomposition schemes promote modular modeling. Some well-defined rules have been set for the interfacing of injectors (or two-ports) with the network and Satellite to the Central sub-domain. Thus, as long as these rules are followed, the power system models used in RAMSES are modular. That is, components can be easily added, removed, or combined to form complex models without sacrificing the simulation performance.

A tool is provided for the user to program new models that can be included in RAMSES. Figure C.4 shows the procedure for introducing user-defined models to the simulator. At the moment of writing this thesis, the following four types of models are supported: torque control of synchronous machine, excitation control of synchronous machine, injector, and twoports.

Apart from the components described above, there is a separate class of components called Discrete-time ConTroLlers (DCTLs). These controllers act at discrete times, when a condition is fulfilled, or at multiple of their internal activation period. Their actions are applied after the simulation time step is completed. Examples of such DCTLs are the LTCs, ASRTs, and MPC-based DNV controllers, seen in Chapters 4 and 5.

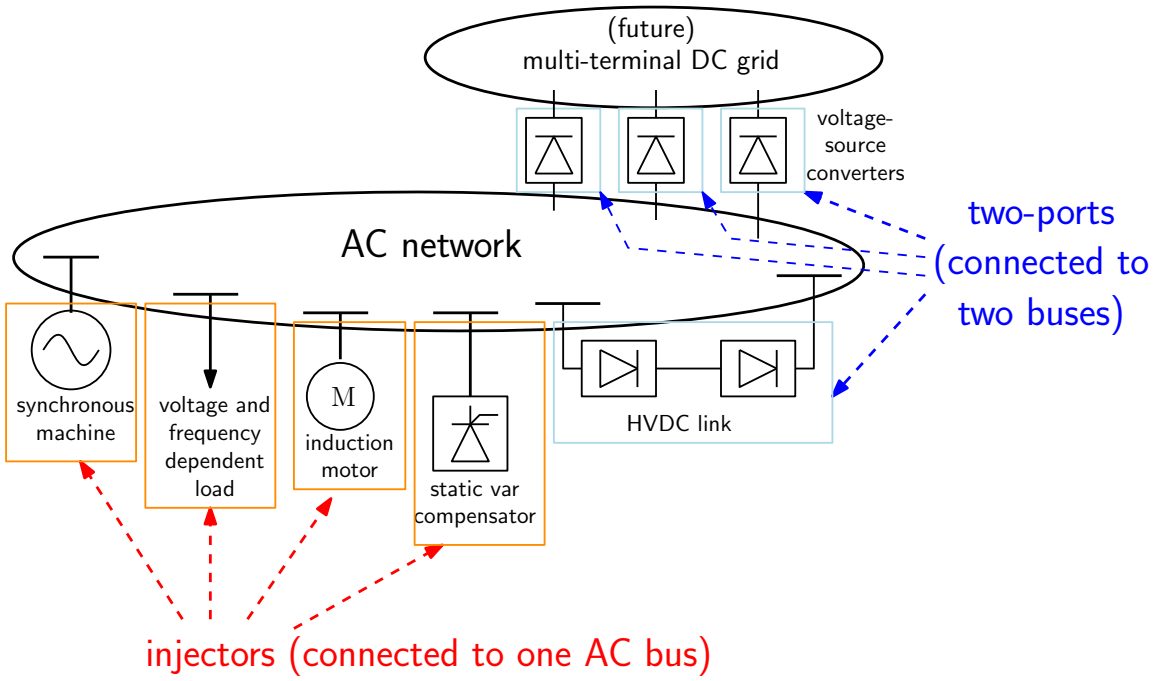


Figure C.2: Overview of power system components in RAMSES

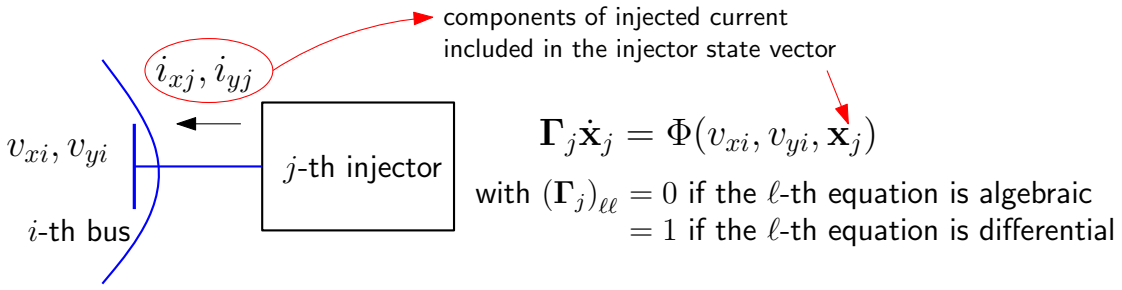


Figure C.3: Injector interface with network in RAMSES

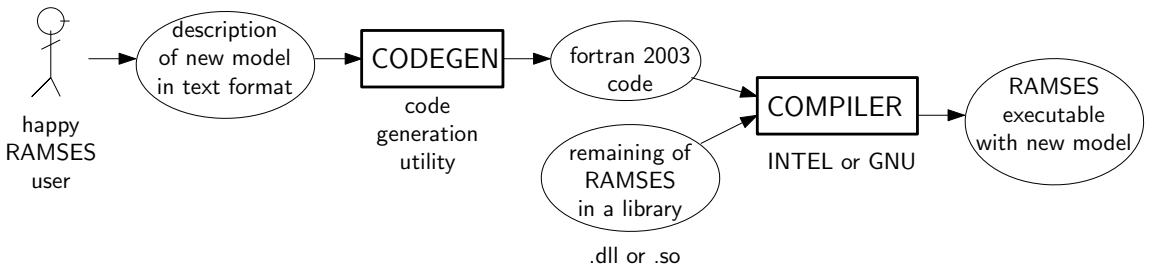


Figure C.4: Procedure for introducing new model in RAMSES

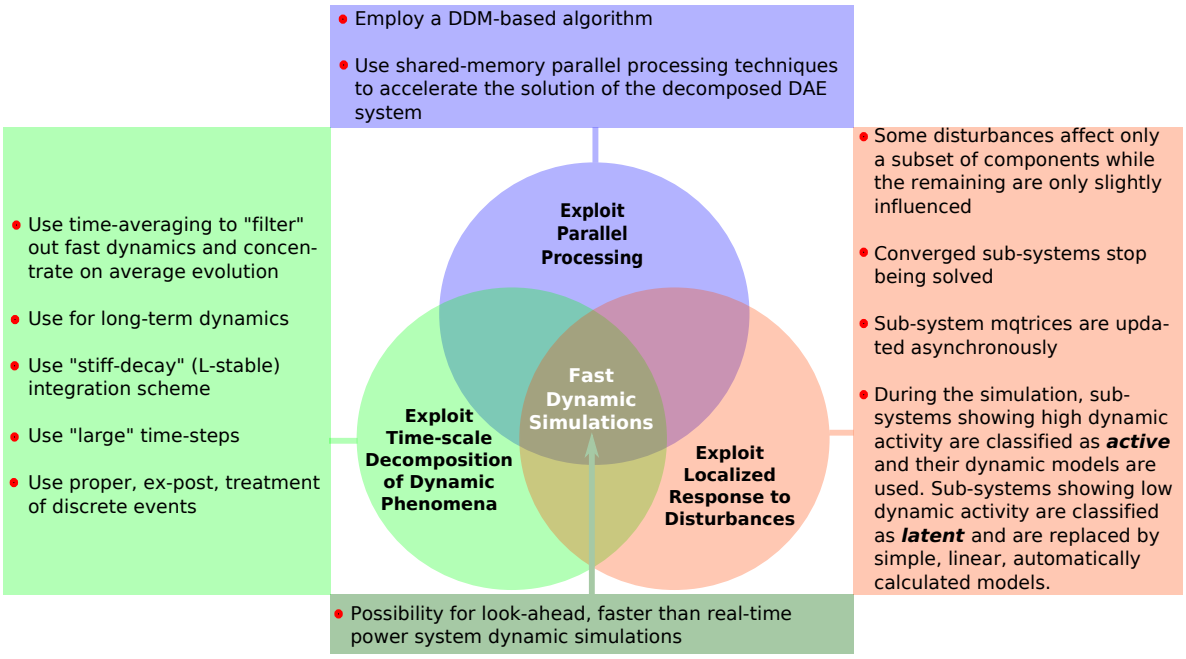


Figure C.5: Acceleration techniques used in RAMSES

C.3 Acceleration techniques

Three groups of acceleration techniques, sketched in Fig. C.5, are used to speedup the simulation procedure. The first one is the parallelization of the DDM-based algorithm to exploit multi-core computers and provide computational speedup. The second acceleration technique exploits localization. The last acceleration technique exploits the possibility of time-scale decomposition of dynamic phenomena. The first two techniques have been presented and discussed in Chapters 4 and 5. The third acceleration technique is summarized below.

C.3.1 Time-scale decomposition

When considering long-term dynamic simulations (i.e. for long-term voltage stability), some fast components of the response may not be of interest and could be partially or totally omitted to provide faster simulations. This can be achieved either through the use of simplified models [VGL06] or with a dedicated solver applying time-averaging [FCPV11].

While model simplification offers a big acceleration with respect to detailed simulation, some drawbacks exist. First, the separation of slow and fast components might not be possible for complex or black-box models. Furthermore, there is a need to maintain both detailed and simplified models. Finally, if both short and long-term evolutions are of interest, simplified and detailed simulations must be properly coupled [VGL06].

At the same time, solvers using “stiff-decay” (L-stable) integration methods, such as BDF, with large enough time-steps can discard some fast dynamics. Such a solver, applied on a detailed model, can “filter” out the fast dynamics and concentrate on the average evolution

of the system. The most significant advantage of this approach is that it processes the detailed, referenced model. Furthermore, this technique allows combining detailed simulation for short-term by limiting the time-step size, and time-averaged long-term by increasing it. This time-averaging technique, proposed in [FCPV11], is available to be used in RAMSES for the discretization of the DAEs. However, in the results shown in this thesis, this technique was not used. That is, a small and constant time-step size was used to allow for the assessment of the acceleration provided by the other two techniques only.

C.4 Software implementation

RAMSES implements the algorithms detailed in Chapters 4 and 5, as well as the integrated VDHN scheme described in Section 1.2.5.2. The programming language is Fortran 2003 and the OpenMP API has been used for the parallelization of the algorithms. The sparse linear solver HSL MA41 is used for the solution of the sparse matrices and the Intel MKL Lapack libraries for the solution of the dense matrices with the DGETRF and DGETRS subroutines. The sequential versions of the aforementioned libraries are used, as their parallelized counterparts introduced increased OHC and led to slow-downs of the simulation.

The code has been compiled and successfully tested on several operating systems (Windows 7, 8, 8.1, Linux Debian 6, Linux Ubuntu 14.10, etc.). Moreover, the code is compatible with GNU and Intel Fortran compilers. Although, Intel Fortran compiler has shown higher performance than GNU, especially when executing in parallel.

C.4.1 Why Fortran?

"*I don't know what the language of the year 2000 will look like, but I know it will be called Fortran.*"
—Tony Hoare, winner of the 1980 Turing Award.

Fortran is, for better or worse, one of the main programming languages specifically designed for scientific numerical computing. It has advanced array handling functions, with succinct array operations on both whole arrays and slices, comparable with MATLAB or Python Numpy, but much faster. The language is carefully maintained with speed of execution in mind. For example, pointers are restricted in such a way that it is immediately obvious if there might be aliasing. It has advanced support for shared-memory parallel computing (through OpenMP), distributed-memory parallel computing (through co-arrays and MPI), and vectorization.

Fortran has a huge long tradition, and this can be considered as an advantage and disadvantage together. On the one hand, there is a plethora of great libraries written in Fortran (with BLAS and LAPACK as examples). On the other hand, it comes with much historical baggage targeting to keep backward compatibility. Nevertheless, modern Fortran implementations have little to envy from other languages.

When the implementation performs a lot of number crunching, Fortran remains one of the top choices. That is the reason why many of the most sophisticated simulation codes running at supercomputing centers around the world are still written in it. On the contrary, Fortran would be a terrible language to write a web browser, perform communication tasks, manipulate databases, etc. To each task its tool.

The core of RAMSES, which performs the numerical simulation, is written in Fortran for increased performance. However, using the C-interoperability functions of Fortran, several interfaces have been developed to facilitate its use.

C.4.2 Command line

The most basic interface provided by RAMSES is the command line mode. The user executes the program and can provide the necessary files needed for the simulation (test-system description, contingency description, etc.) in an interactive way or in batch execution. This interface is useful for remote execution on systems without graphic interfaces and to be embedded in scripts as part of more complex procedures.

C.4.3 Dynamic library

RAMSES is also provided as a dynamic library (.dll in Windows or .so in Linux). This option allows the user to include the simulator in his own software or load it in a script (e.g. Python, Perl, etc.). The connection to the library is performed through C-type calls and the user can start, stop, pause, or modify the simulation. Moreover, the calling software has access to all the outputs of the simulator (through “get” subroutines) and can take decisions based on the dynamic response of the simulation.

C.4.4 Graphic user interface

A Java-based Graphic User Interface was developed that allows for an easy-to-use and standalone execution of the simulator. Moreover, the JAVA language provides compatibility with multiple platforms (Linux and Windows). The software is provided in a single JAVA archive (.jar file), including all necessary executables and libraries to perform simulations and visualize results. Thus, the software is ready to be used with no installation.

C.4.5 MATLAB

RAMSES can interface with MATLAB in two ways. First, the RAMSES dynamic library can be embedded in MATLAB scripts as part of more complex processes. These processes can include the preparation of the dynamic data to be used for the simulations as well as post processing and control action to be taken based on the simulation output. In this case, MATLAB is the simulation driver and RAMSES is used a fast dynamic simulation library.

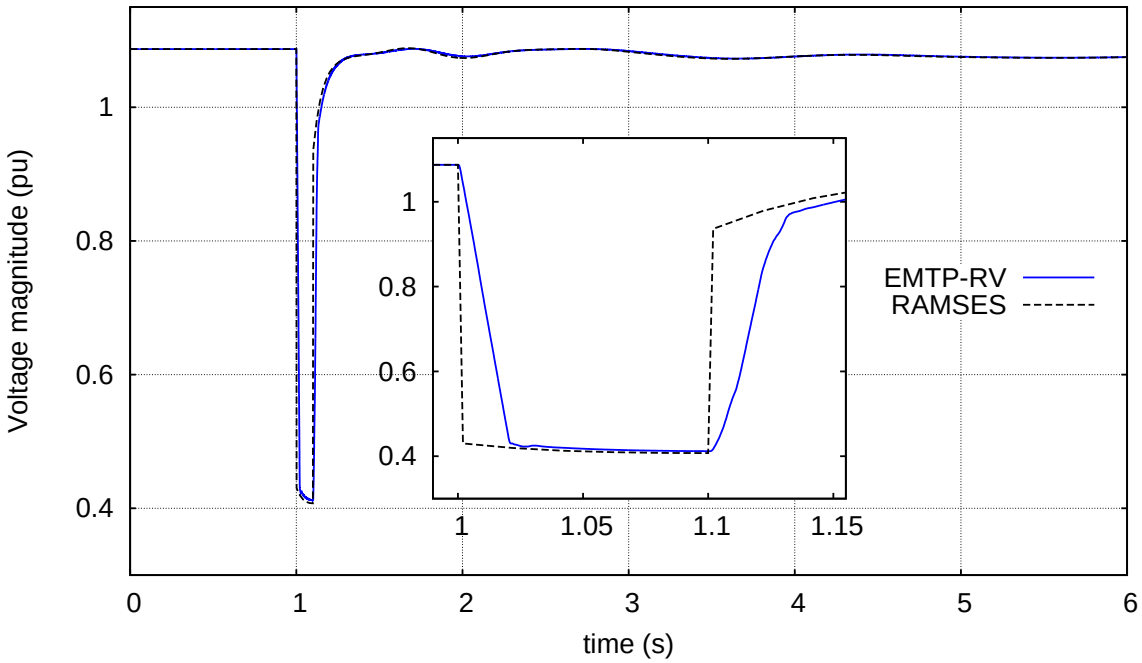


Figure C.6: Scenario 1: Voltage evolution at bus 4043

In addition, RAMSES can interface with MATLAB using the *MATLAB Engine* mechanisms, for rapid prototyping of power system components and discrete controllers (DCTLs). In this case, RAMSES acts as the simulation driver and MATLAB is called to compute the response of the components being developed or tested.

C.5 Validation

The dynamic response of RAMSES was validated against the well-known power system dynamic simulation software EMTP-RV. The latter is a EMT type software while RAMSES uses the phasor approximation (see Section 1.2). In general, EMT software are more accurate than phasor mode ones as more detailed models are used and the full wave is simulated, rather than phasors rotating at the nominal frequency [Yan14]. However, EMT simulations are more time consuming and the power systems being studied are usually limited in size.

Two scenarios, based on the Nordic system presented in Section 1.3, were used to compare the response of the two software. For the comparison of the bus voltage evolution, the fundamental frequency (50 Hz) wave was extracted from EMTP-RV. The simulations were performed on a multi-core laptop computer (Machine 3) and the single-level, parallel algorithm of Chapter 4 was used for RAMSES. The results shown in this appendix were part of the development of a co-simulation algorithm [PAGV14]. A journal paper has been submitted[1].

[1] F. Plumier, P. Aristidou, C. Geuzaine, T. Van Cutsem, "Co-simulation of Electromagnetic Transients and Phasor Models: a Relaxation Approach", Submitted to IEEE Transactions on Power Delivery, 2015.

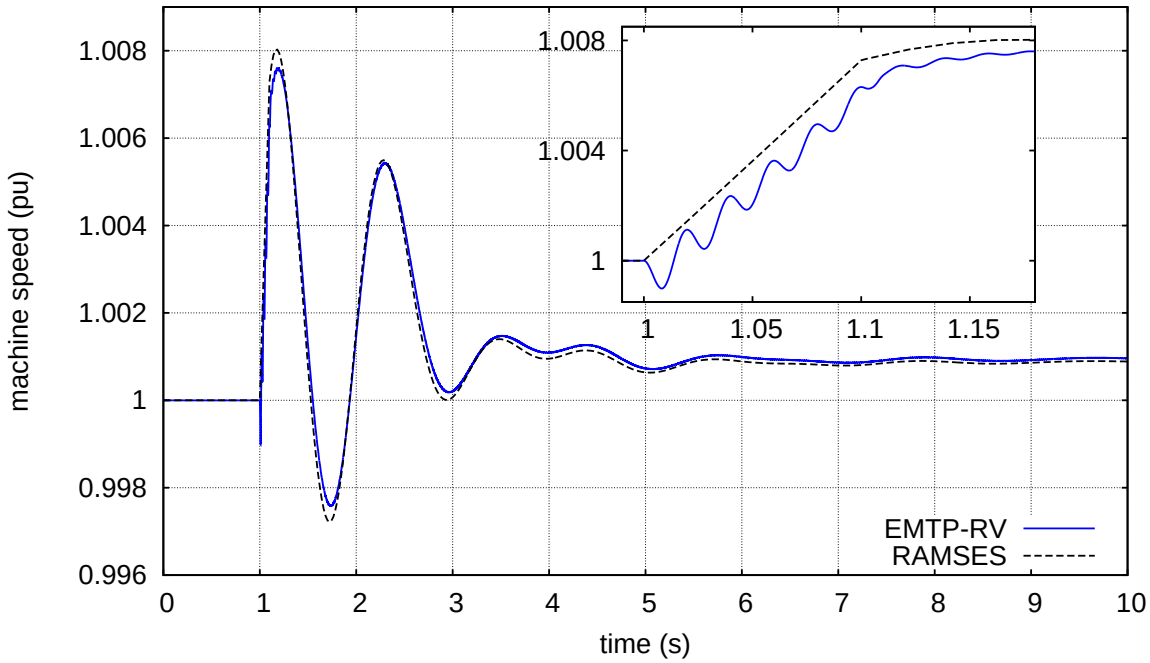


Figure C.7: Scenario 1: Machine speed evolution at generator $g15b$

C.5.1 Scenario 1

The disturbance of concern is a three-phase solid fault at $t = 1$ s on line $4046 - 4047$, near bus 4047, lasting five cycles (at 50 Hz) and cleared by opening the line, which remains opened. Following, the system is simulated over an interval of 10 s. In RAMSES, a time-step size of one cycle is used, while in EMTP-RV of 100 μs.

Figure C.6 shows the voltage evolution on transmission bus 4043 computed from both software. In the zoom of the same figure, it can be seen that the bus voltage in RAMSES drops directly after the fault, while in EMTP-RV there is a time constant. This is because RAMSES, as a phasor mode simulator, uses only algebraic equations to model the network, while EMTP-RV a more accurate, differential model. Nevertheless, the dynamic response of RAMSES matches EMTP-RV.

Figure C.7 shows the machine speed of generator $g15b$. Disregarding some higher frequency components in EMTP-RV (which are not captured by phasor mode simulations), the two curves match.

C.5.2 Scenario 2

The disturbance of concern is a three-phase solid fault at $t = 1$ s on line $1042 - 1044$, near bus 1042, lasting 10.5 cycles (at 50 Hz) and cleared by opening the line, which remains opened. Following, the system is simulated over an interval of 10 s. In RAMSES, a time-step size of one cycle is used, while in EMTP-RV of 100 μs.

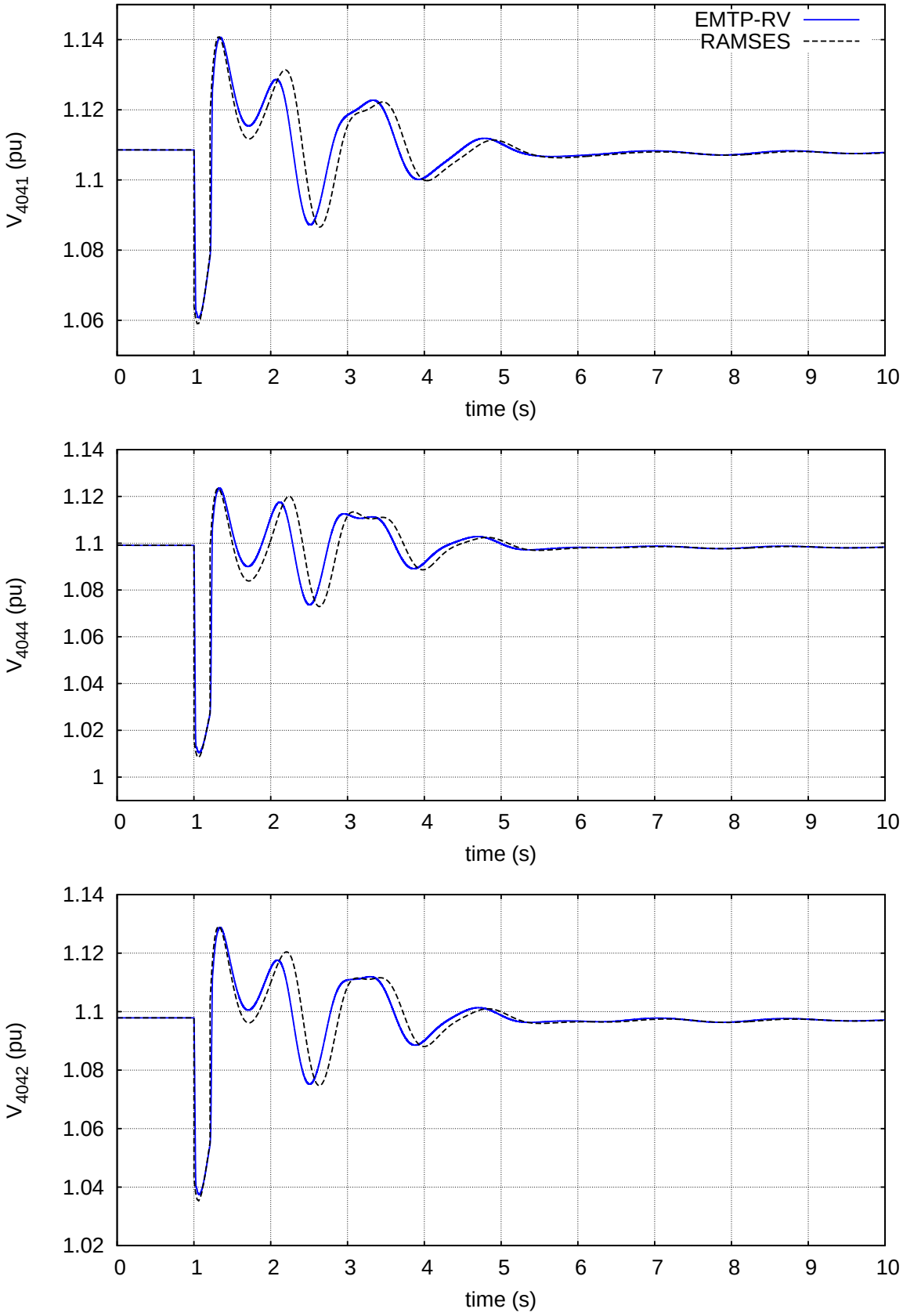


Figure C.8: Scenario 2: Voltage evolution at three buses

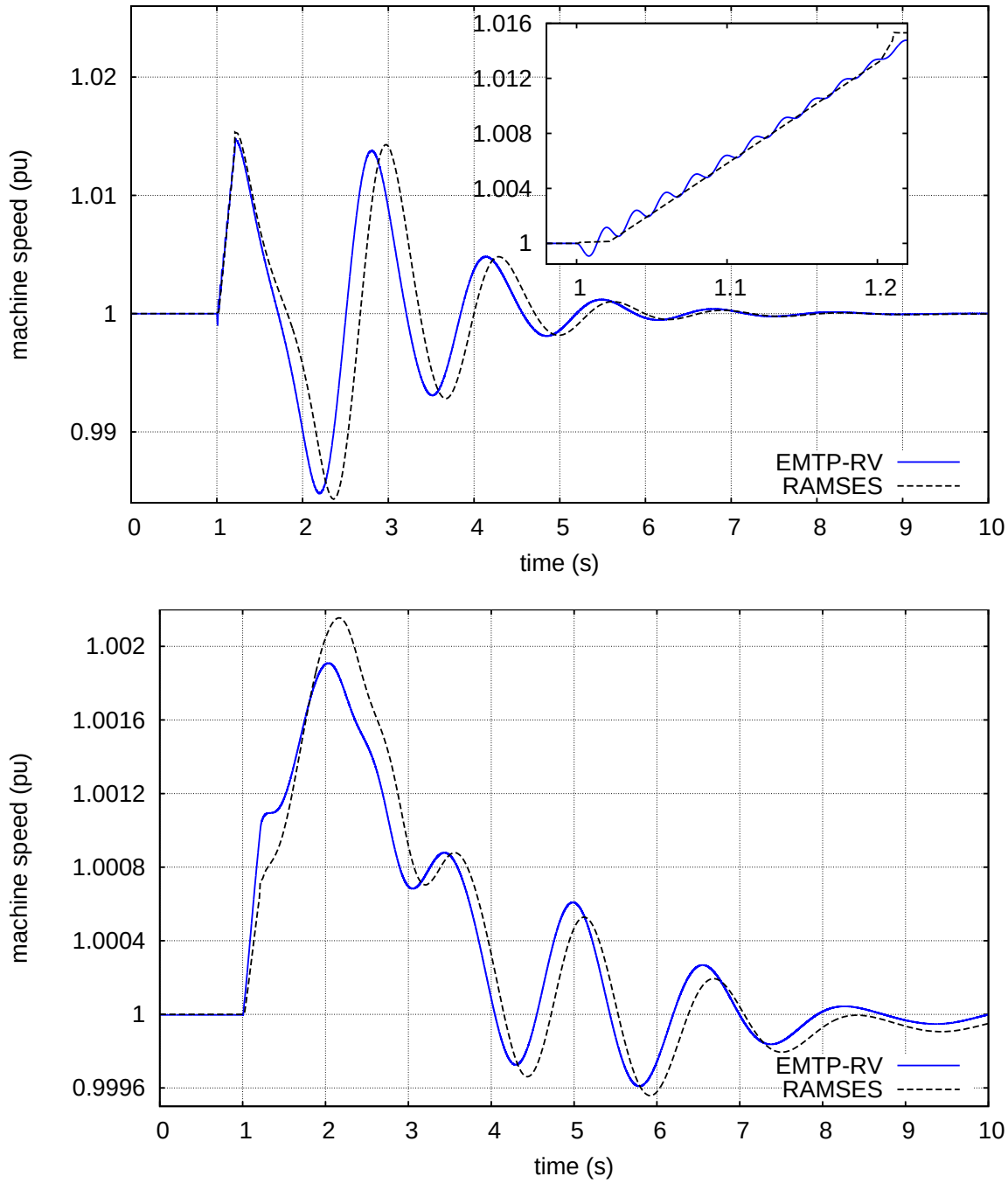


Figure C.9: Scenario 2: Machine speed evolution at generators $g6$ and $g18$

Figure C.8 shows the voltage evolution on three transmission buses computed from both software. It can be seen that the dynamic response of RAMSES matches EMTP-RV.

Figure C.9 shows the machine speed of generators $g6$ and $g18$. Disregarding some higher frequency components in EMTP-RV (which are not captured by phasor mode simulations), the two curves match.

APPENDIX **D**

Numerical profiling

In this appendix, two example numerical profilings of the results presented in Sections 4.9 and 5.8 are shown. These profilings are representative of how the localization techniques modify the number of operations in the DDM algorithms and are important to understand their performance.

Table D.1 shows the number of operations performed, over all time steps and iterations for the test-case of Section 4.9.2. Since the DDM used is not a relaxation technique but a direct method, the number of operations remain the same either in parallel or sequential execution. As shown in Section 4.7, Config. I is equivalent to the integrated and, since the same matrix update and solution criteria are used for both algorithms, the number of matrix updates and solutions are the same. Thus, the integrated method performs approximately 1054 integrated Jacobian updates and factorizations and 23886 system solutions. Some deviations might exist due to numerical differences of the solvers used. That is, in the integrated method, the entire system is solved with the sparse solver HSL MA41, while in the DDM the injectors are solved using the Intel MKL Lapack routines.

Comparing Config. I with Config. II and III, it can be seen that in the latter two, the number of reduced matrix updates and solutions significantly decreases due to the asynchronous update (Section 4.6.2) and the skipping of converged sub-systems (Section 4.6.1). The same observation is true for the injector system updates, factorizations and Schur-complement term computation as well as for the injector system solutions.

On the other hand, the number of reduced system Right-Hand Side (RHS) evaluations increases in Config. II and III. The reason for this is that the localization techniques disturb the convergence rate of the algorithm as described in Section 4.7. Thus, when the localization techniques are used, the algorithm requires on average *more iterations per time instant to converge*; although, each iteration is computationally *much cheaper* than without the localization techniques. Nevertheless, the number of reduced system RHS evaluations increases proportionally to the total number of iterations as this is needed to ensure and check the convergence of the algorithm.

Table D.1: Chapter 4, HQ: Numerical profiling of Algorithm 4.3 in test-case of Section 4.9.2

Configuration	I	II	III ($\epsilon_L = 0.1$ MVA)
Speedup ($M = 1$)	1.3	1.6	3.3
Reduced system (4.11) updates and factorizations	1054	30	30
Reduced system (4.11) solutions	23886	12568	10731
Reduced system (4.11) RHS evaluations	38591	47530	44828
Injector system (4.8) updates, factorizations, and Schur-complement computations	4849454 (1054)[1]	591688 (129)	607109 (131)
Injector system (4.8) solutions	109899486 (23886)	72373575 (15730)	21417786 (4654)[2]
Injector system (4.8) RHS evaluations	287148410 (62410)	290750838 (63193)	83679539 (18183)

Finally, the number of injector system RHS evaluations increases in Config. II and decreases in III. For the first increase, the reason is the same as above; the higher number of iterations leads to proportionally more RHS evaluations to guarantee the algorithm's convergence. However, in Config. III, several of the injector models are replaced by the linear equivalents as described in Section 4.6.3. Since these models are linear, their RHS after each iteration solution is zero, thus there is no need to be computed.

Table D.2 provides the same information over all time steps and iterations for the test-case of Section 5.8.1.2, using the two-level DDM. The integrated method performs approximately the same integrated Jacobian updates and system solutions as the global reduced system of Config. I. In reality, the integrated scheme needs more iterations per time instant to reach the desired convergence accuracy. The reason relates to the selection of the base power as explained in Section 5.3.4. That is, the integrated scheme scheme uses the smaller base power $S_{base} = 2$ MVA for the entire system to ensure the accuracy of the Satellite sub-systems, while the two-level DDM uses $S_{baseC} = 100$ MVA and $S_{baseS} = 2$ MVA.

Comparing Config. I with Config. II and IIIa, it can be seen that in the latter two, the number of global and satellite reduced matrix updates and solutions are fewer due to the asynchronous update and the skipping of converged sub-systems. Similarly for the injector system updates, factorizations and Schur-complement term computation as well as for the injector system solutions.

However, the reduced and injector system updates are more in Config. IIIa than II. Whenever a Satellite sub-domain or injector is switched from active to latent, an update of its

[1] In parenthesis is the average value per injector (N=4601).
[2] Solutions of the linear equivalent systems not counted.

Table D.2: Chapter 5, Nordic variant 1, Scenario 2b: Numerical profiling of Algorithm 5.4 in test-case of Section 5.8.1.2

Configuration	I	II	IIIa
Speedup ($M = 1$)	0.9	1.1	2.9
Global reduced system (5.15) updates and factorizations	715	54	58
Global reduced system (5.15) solutions	24945	11869	10068
Global reduced system (5.15) RHS evaluations	37276	38724	34553
Satellite reduced system (5.14) updates and factorizations	104390 (715)[3]	7225 (50)	10492 (72)
Satellite reduced system (5.14) solutions	3641970 (24945)	1764638 (12087)	863299 (5913)[4]
Satellite reduced system (5.14) RHS evaluations	5442296 (37276)	5653704 (38724)	5044738 (34553)
Injector system updates, factorizations, and Schur-complement computations	14734005 (714)	1340581 (65)	1877431 (91)
Injector system solutions	514041615 (24943)	271106152 (13155)	69548423 (3374)
Injector system RHS evaluations	1279715307 (62097)	1066247854 (51739)	259914368 (12612)

matrices is forced to ensure the use of the most accurate linear equivalent model possible. Thus, in cases with increased number of switches (many marginally latent components), the number of matrix updates and factorizations might increase compared to Config. II. Nevertheless, the significantly fewer number of Satellite reduced system and injector solutions, compensates for this increase and provides speedup.

Finally, the number of Global and Satellite reduced system RHS evaluations increases in Config. II and decreases in IIIa. For the first increase, when the localization techniques are used, the algorithm requires on average more iterations per time instant to converge, thus more RHS evaluations to guarantee the algorithm's convergence. However, in Config. IIIa, several of the Satellite sub-domain models are replaced by the linear equivalents and their RHS after each iteration solution is zero.

[3] In parenthesis is the average value per DN (L=146).
[4] Solutions of the linear equivalent systems not counted.

Bibliography

[ABLS97] G. Aloisio, M. Bochicchio, M. La Scala, and R. Sbrizzai, "A distributed computing approach for real-time transient stability analysis," *IEEE Transactions on Power Systems*, vol. 12, no. 2, pp. 981–987, May 1997. 2.4.2

[ADK12] D. Ablakovic, I. Dzafic, and S. Kecici, "Parallelization of radial three-phase distribution power flow using GPU," in *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Berlin, Oct. 2012. 2.4.1

[AF12] S. Abhyankar and A. J. Flueck, "Real-time power system dynamics simulation using a parallel Block-Jacobi preconditioned Newton-GMRES scheme," in *Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012*, Salt Lake City, Nov. 2012, pp. 299–305. 2.4.3, 3.3.1

[AFV13a] P. Aristidou, D. Fabozzi, and T. Van Cutsem, "Exploiting Localization for Faster Power System Dynamic Simulations," in *Proc. of 2013 IEEE PES PowerTech conference*, Grenoble, Jun. 2013. 1.5, 4.4

[AFV13b] ——, "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, Oct. 2013. 1.5

[AFV14] ——, "A Schur Complement Method for DAE Systems in Power System Dynamic Simulations," in *Domain Decomposition Methods in Science and Engineering XXI*, ser. Lecture Notes in Computational Science and Engineering, J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. Widlund, Eds. Springer International Publishing, 2014, vol. 98, pp. 719–727. 1.5

[Ait87] P. W. Aitchison, "Diakoptics as a general approach in engineering," *Journal of Engineering Mathematics*, vol. 21, no. 1, pp. 47–58, 1987. 3.3.3

[Alv79] F. Alvarado, “Parallel Solution of Transient Problems by Trapezoidal Integration,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-98, no. 3, pp. 1080–1090, May 1979. 3.3.3

[ANL+12] U. D. Annakkage, N. K. C. Nair, Y. Liang, A. M. Gole, V. Dinavahi, B. Gustavsen, T. Noda, H. Ghasemi, A. Monti, M. Matar, R. Iravani, and J. A. Martinez, “Dynamic system equivalents: A survey of available techniques,” *IEEE Transactions on Power Delivery*, vol. 27, no. 1, pp. 411–420, 2012. 5.1

[ASC13] S. Abhyankar, B. Smith, and E. Constantinescu, “Evaluation of overlapping restricted additive schwarz preconditioning for parallel solution of very large power flow problems,” in *Proceedings of the 3rd International Workshop on High Performance Computing, Networking and Analytics for the Power Grid - HiPCNA-PG ’13*, New York, 2013. 2.4.3, 3.3.1

[AT90] A. Abur and P. Tapadiya, “Parallel state estimation using multiprocessors,” *Electric Power Systems Research*, vol. 18, no. 1, pp. 67–73, Jan. 1990. 2.4.2

[AV13] P. Aristidou and T. Van Cutsem, “Dynamic simulations of combined transmission and distribution systems using decomposition and localization,” in *Proceedings of IEEE PES 2013 PowerTech conference*, Grenoble, 2013. 1.5, 4.4

[AV14a] ——, “Dynamic Simulations of Combined Transmission and Distribution Systems using Parallel Processing Techniques,” in *Proceedings of 18th Power System Computational Conference (PSCC)*, Warsaw, 2014. 1.5

[AV14b] ——, “Algorithmic and computational advances for fast power system dynamic simulations,” in *Proc. of 2014 IEEE PES General Meeting*, Washington DC, Jul. 2014. 1.5, 4.4

[AV14c] ——, “Parallel Computing and Localization Techniques for Faster Power System Dynamic Simulations,” in *Proceedings of 2014 CIGRE Belgium conference*, Brussels, 2014. 1.5

[AV15] ——, “A parallel processing approach to dynamic simulations of combined transmission and distribution systems,” *International Journal of Electrical Power & Energy Systems*, vol. 72, pp. 58–65, 2015. 1.5

[AZS96] M. Amano, A. Zecevic, and D. Siljak, “An improved block-parallel Newton method via epsilon decompositions for load-flow calculations,” *IEEE Transactions on Power Systems*, vol. 11, no. 3, pp. 1519–1527, 1996. 2.4.2

[BBMP05] P. Biskas, A. Bakirtzis, N. Macheras, and N. Pasialis, “A Decentralized Implementation of DC Optimal Power Flow on a Network of Computers,” *IEEE Transactions on Power Systems*, vol. 20, no. 1, pp. 25–33, Feb. 2005. 2.4.2

[BCP95] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, Jan. 1995. 1.2.2, 1.2.2.1, 1.2.2.1, 1.2.2.1, 1.2.2.1, 1.2.2.1, 1.2.2.1, 1.2.3, 1.2.5.1

[BDM73] C. G. Broyden, J. E. Dennis, and J. J. Moré, "On the Local and Superlinear Convergence of Quasi-Newton Methods," *IMA Journal of Applied Mathematics*, vol. 12, no. 3, pp. 223–245, 1973. 4.7, A.1

[BDP96] K. Burrage, C. Dyke, and B. Pohl, "On the performance of parallel waveform relaxations for differential systems," *Applied Numerical Mathematics*, vol. 20, no. 1-2, pp. 39–55, Feb. 1996. 3.3.3

[BHSt13] A. R. Brodtkorb, T. R. Hagen, and M. L. Sæ tra, "Graphics processing unit (GPU) programming strategies and trends in GPU computing," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 4–13, Jan. 2013. 2.4.1

[BIG76] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses (Theory And Applications)*. Krieger, 1976. 4.4

[BK99] Z. Bartoszewski and M. Kwapisz, "On the Convergence of Waveform Relaxation Methods for Differential-Functional Systems of Equations," *Journal of Mathematical Analysis and Applications*, vol. 235, no. 2, pp. 478–496, Jul. 1999. 4, 3.3.3

[Bos95] 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. 2.4.3

[Bra93] V. Brandwajn, "Localization Concepts in (In)-Security Analysis," in *Proceedings of 1993 Joint International Power conference PowerTech*, Athens, 1993. 4.6

[Bre74] R. P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," *Journal of the ACM*, vol. 21, no. 2, pp. 201–206, Apr. 1974. 2.5.4

[Bro70] C. G. Broyden, "The Convergence of Single-Rank Quasi-Newton Methods," *Mathematics of Computation*, vol. 24, no. 110, p. 365, Apr. 1970. 4.7, A.1

[BTS96] S. Bernard, G. Trudel, and G. Scott, "A 735 kV shunt reactors automatic switching system for Hydro-Quebec network," *IEEE Transactions on Power Systems*, vol. 11, no. 4, pp. 2024–2030, 1996. 1.3.2

[BVL95] A. Bose, A. Valette, and F. Lafrance, "Parallel implementation of power system transient stability analysis," *IEEE Transactions on Power Systems*, vol. 10, no. 3, pp. 1226–1233, 1995. 2.4.3

[BWJ+12] W. Baijian, G. Wenxin, H. Jiayi, W. Fangzong, and Y. Jing, "GPU based parallel simulation of transient stability using symplectic Gauss algorithm and preconditioned GMRES method," in *Proceedings of 2012 Power Engineering and Automation (PEAM) conference*, Wuhan, Sep. 2012. 2.4.1

[Cat04] E. Catinas, "The inexact, inexact perturbed, and quasi-Newton methods are equivalent models," *Mathematics of Computation*, vol. 74, no. 249, pp. 291–302, Mar. 2004. A.2

[CB93] J. Chai and A. Bose, "Bottlenecks in parallel algorithms for power system stability analysis," *IEEE Transactions on Power Systems*, vol. 8, no. 1, pp. 9–15, Feb. 1993. 2.4.2, 2.5.1, 3.3.2

[CDC02] K. Chan, R. Dai, and C. Cheung, "A coarse grain parallel solution method for solving large set of power systems network equations," in *Proceedings of 2002 International Conference on Power System Technology (PowerCon)*, vol. 4, no. 1, Kunming, 2002, pp. 2640–2644. 2.4.2, 3.3.2

[Cel06] F. Cellier, *Continuous System Simulation*. Boston: Kluwer Academic Publishers, 2006. 1.2.4

[Cha95] K. Chan, "Efficient heuristic partitioning algorithm for parallel processing of large power systems network equations," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 142, no. 6, p. 625, 1995. 3.3.1

[Cha01a] ——, "Parallel algorithms for direct solution of large sparse power system matrix equations," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 148, no. 6, p. 615, 2001. 2.4.2, 3.3.2

[Cha01b] R. Chandra, *Parallel programming in OpenMP*. Morgan Kaufmann, 2001. 2.6.3

[CI90] M. Crow and M. Ilic, "The parallel implementation of the waveform relaxation method for transient stability simulations," *IEEE Transactions on Power Systems*, vol. 5, no. 3, pp. 922–932, Aug. 1990. 2.4.2, 3.3.3, 3.3.3

[Cie13] S. Cieslik, "GPU Implementation of the Electric Power System Model for Real-Time Simulation of Electromagnetic Transients," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, Paris, 2013. 2.4.1

[CIW89] M. Crow, M. Ilic, and J. White, "Convergence properties of the waveform relaxation method as applied to electric power systems," in *Proceedings of 1989 IEEE International Symposium on Circuits and Systems*, no. 4, 1989, pp. 1863–1866. 3.3.3, 3.3.3

[CJV07] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007. 2.6.2, 2.6.2, 2.6.3

[CM11] T. Cadeau and F. Magoules, "Coupling Parareal and Waveform Relaxation methods for power systems," in *Proceedings of 2011 International Conference on Electrical and Control Engineering (ICECE)*, no. 4, Yichang, Sep. 2011, pp. 2947–2950. 3.3.3

[CRTT11] CRSA, RTE, TE, and TU/e, "D4.1: Algorithmic requirements for simulation of large network extreme scenarios," Tech. Rep., 2011. [Online]. Available: http://www.fp7-pegase.eu/ 4, 3.3.1, 3.3.3

[CZBT91] J. Chai, N. Zhu, A. Bose, and D. 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. 2.4.3, 3.3.2

[DES82] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact newton methods," *SIAM Journal on Numerical Analysis*, vol. 19, no. 2, pp. 400–408, 1982. A.2, A.2, A.2

[DFK92] I. Decker, D. Falcao, and E. Kaszkurewicz, "Parallel implementation of a power system dynamic simulation methodology using the conjugate gradient method," *IEEE Transactions on Power Systems*, vol. 7, no. 1, pp. 458–465, 1992. 2.4.2

[DFK96] ——, "Conjugate gradient methods for power system dynamic simulation on parallel computers," *IEEE Transactions on Power Systems*, vol. 11, no. 3, pp. 1218–1227, 1996. 2.4.2

[DGF12] J. K. Debnath, A. M. Gole, and S. Filizadeh, "Electromagnetic transient simulation of large-scale electrical power networks using graphics processing units," in *Proceedings of 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Montreal, Apr. 2012. 2.4.1

[DM74] J. E. Dennis and J. J. Moré, "A characterization of superlinear convergence and its application to quasi-Newton methods," *Mathematics of Computation*, vol. 28, no. 126, pp. 549–549, May 1974. 4.7, A.1

[DN12] I. Dzafic and H. T. Neisius, "Quasi-parallel network applications in real-time distribution management system," *International Journal of Innovative Computing and Applications*, vol. 4, no. 1, p. 3, 2012. 2.4.3

[DS72] H. Dommel and N. Sato, "Fast Transient Stability Soultions," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-91, no. 4, pp. 1643–1650, Jul. 1972. 1.2.5

[DS11] H. Dag and G. Soykan, "Power flow using thread programming," in *Proceedings of 2011 IEEE PES PowerTech conference*, Trondheim, Jun. 2011. 2.4.3

[DW84] J. Dennis and H. Walker, "Inaccuracy in quasi-Newton methods: Local improvement theorems," in *Mathematical Programming at Oberwolfach II*, B. Korte and K. Ritter, Eds. Springer Berlin Heidelberg, 1984, vol. 22, pp. 70–85. A.1

[Eck13] A. Eckner, "Algorithms for Unevenly-Spaced Time Series: Moving Averages and Other Rolling Operators," Tech. Rep., 2013. [Online]. Available: www.eckner.com/papers/ts_alg.pdf 4.6.3.2

[EKM08] M. El-Kyal and A. Machmoum, "Superlinear convergence of asynchronous multi-splitting waveform relaxation methods applied to a system of nonlinear ordinary differential equations," *Mathematics and Computers in Simulation*, vol. 77, no. 2-3, pp. 179–188, Mar. 2008. 3.3.3

[EKM+11] A. Ellis, Y. Kazachkov, E. Muljadi, P. Pourbeik, and J. J. Sanchez-Gasca, "Description and technical specifications for generic WTG models - A status report," in *Proceedings of 2011 IEEE PES Power Systems Conference and Exposition (PSCE 2011)*, Phoenix, Mar. 2011. 1.3.1

[ES13] M. Eremia and M. Shahidehpour, *Handbook of Electrical Power System Dynamics: Modeling, Stability, and Control*. John Wiley & Sons, 2013. 1.2.2.1

[Fab12] D. Fabozzi, "Decomposition, Localization and Time-Averaging Approaches in Large-Scale Power System Dynamic Simulation," Ph.D. dissertation, Univeristy of Liege, 2012. 1.2.1, 1.2.2.1, 1.2.3, 1.2.4, 1.2.5.2

[FCHV13] D. Fabozzi, A. S. Chieh, B. Haut, and T. Van Cutsem, "Accelerated and Localized Newton Schemes for Faster Dynamic Simulation of Large Power Systems," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4936–4947, Nov. 2013. 1.2.5.2, 4.1

[FCPV11] D. Fabozzi, A. S. Chieh, P. Panciatici, and T. Van Cutsem, "On simplified handling of state events in time-domain simulation," in *Proc. of 17th Power System Computational Conference (PSCC)*, Stockholm, 2011. 1.2.4, C.3.1

[FKA93] D. Falcao, E. Kaszkurewicz, and H. Almeida, "Application of parallel processing techniques to the simulation of power system electromagnetic transients," *IEEE Transactions on Power Systems*, vol. 8, no. 1, pp. 90–96, 1993. 2.4.2

[Flu02a] A. Flueck, "A message-passing distributed-memory Newton-GMRES parallel power flow algorithm," in *Proceedings of 2002 IEEE PES Summer Meeting*, Chicago, 2002, pp. 1477–1482. 2.4.2

[Flu02b] ——, “A message-passing distributed-memory parallel power flow algorithm,” in *Proceedings of 2002 IEEE PES Winter Meeting*, vol. 1, New York, 2002, pp. 211–216. 2.4.2

[Fly72] M. J. Flynn, “Some Computer Organizations and Their Effectiveness,” *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 948–960, Sep. 1972. 2.3.1

[FM04] J. Fung and S. Mann, “Using multiple graphics cards as a general purpose parallel computer: applications to computer vision,” in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004)*, Cambridge, 2004, pp. 805–808. 2.4.1

[FMT13] C. Fu, J. D. McCalley, and J. Tong, “A Numerical Solver Design for Extended-Term Time-Domain Simulation,” *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4926–4935, Nov. 2013. 1.2.5

[FP78] J. Fong and C. Pottle, “Parallel Processing of Power System Analysis Problems Via Simple Parallel Microcomputer Structures,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-97, no. 5, pp. 1834–1841, Sep. 1978. 2.4.2

[Fra12] F. Franchetti, “A multi-core high performance computing framework for probabilistic solutions of distribution systems,” in *Proceedings of 2012 IEEE PES General Meeting*, San Diego, Jul. 2012. 2.4.3

[FV09] D. Fabozzi and T. Van Cutsem, “Simplified time-domain simulation of detailed long-term dynamic models,” in *Proceedings of 2009 IEEE PES General Meeting*, Calgary, Jul. 2009. 1.2.3, 1.2.6

[FX06] D. Fang and Y. Xiaodong, “A New Method for Fast Dynamic Simulation of Power Systems,” *IEEE Transactions on Power Systems*, vol. 21, no. 2, pp. 619–628, May 2006. 3.3.2

[Gar10] N. Garcia, “Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach,” in *Proceedings of 2010 IEEE PES General Meeting*, Minneapolis, Jul. 2010. 2.4.1

[Gea71] C. W. Gear, *Numerical initial value problems in ordinary differential equations*, prentice-h ed. Prentice-Hall, 1971. 1.2.2.1, 1.2.3

[GHN99] M. Gander, L. Halpern, and F. Nataf, “Optimal convergence for overlapping and non-overlapping Schwarz waveform relaxation,” in *Proceedings of 11th International Conference on Domain Decomposition Methods*, Greenwich, 1999, pp. 27–36. 3.3.3

[GJY+12] C. Guo, B. Jiang, H. Yuan, Z. Yang, L. Wang, and S. Ren, "Performance Comparisons of Parallel Power Flow Solvers on GPU System," in *Proceedings of 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Seoul, Aug. 2012, pp. 232–239. 2.4.1

[GMLT94] G. Granelli, M. Montagna, M. La Scala, and F. Torelli, "Relaxation-Newton methods for transient stability analysis on a vector/parallel computer," *IEEE Transactions on Power Systems*, vol. 9, no. 2, pp. 637–643, May 1994. 2.4.2

[GNV07] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC Power Flow Based Contingency Analysis Using Graphics Processing Units," in *Proceedings of 2007 IEEE PowerTech conference*, Lausanne, Jul. 2007, pp. 731–736. 2.4.1

[GO13] N. Garcia and R. C. Olmos, "GPU-accelerated Poincaré map method for harmonic-oriented analyses of power systems," in *Proceedings of 2013 IEEE PES General Meeting*, Vancouver, 2013. 2.4.1

[Gov10] D. Gove, *Multicore Application Programming: For Windows, Linux, and Oracle Solaris*. Addison-Wesley Professional, 2010. 2.1, 2.5.2, 3, 4.9.4.2

[GSAR09] J. Giri, D. Sun, and R. Avila-Rosales, "Wanted: A more intelligent grid," *IEEE Power and Energy Magazine*, vol. 7, pp. 34–40, 2009. 4.9.4.3

[GSD+03] W. Gao, E. Solodovnik, R. Dougal, G. Cokkinides, and A. Meliopoulos, "Elimination of numerical oscillations in power system dynamic simulation," in *Proceedings of 18th Annual IEEE Applied Power Electronics Conference and Exposition (APEC)*, vol. 2, no. 1, Miami, 2003, pp. 790–794. 1.2.2.1

[GT82] A. Griewank and P. L. Toint, "Local convergence analysis for partitioned quasi-Newton updates," *Numerische Mathematik*, vol. 39, no. 3, pp. 429–448, Oct. 1982. 4.7, A.1

[GTD08] D. Guibert and D. Tromeur-Dervout, "A Schur Complement Method for DAE/ODE Systems in Multi-Domain Mechanical Design," in *Domain Decomposition Methods in Science and Engineering XVII*. Springer, 2008, pp. 535–541. 3.2.1, 3.2.3.1

[Gur88] J. Gurd, "A taxonomy of parallel computer architectures," in *Proceedings of 1988 International Specialist Seminar on the Design and Application of Parallel Digital Processors*, Lisbon, 1988, pp. 57–61. 2.3.2

[GWA11] R. C. Green, L. Wang, and M. Alam, "High performance computing for electric power systems: Applications and trends," in *Proceedings of 2011 IEEE PES General Meeting*, Geneva, 2011. 5.1

[HA11] T. D. Han and T. S. Abdelrahman, "hiCUDA: High-Level GPGPU Programming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 78–90, Jan. 2011. 2.4.1

[Hap74] H. Happ, "Diakoptics- The solution of system problems by tearing," *Proceedings of the IEEE*, vol. 62, no. 7, 1974. 3.3.3

[HB97] L. Hou and A. Bose, "Implementation of the waveform relaxation algorithm on a shared memory computer for the transient stability problem," *IEEE Transactions on Power Systems*, vol. 12, no. 3, pp. 1053–1060, 1997. 2.4.3

[Hed14] W. F. Hederman, "IEEE Joint Task Force on Quadrennial Energy Review," IEEE, Tech. Rep., 2014. 2.5.3

[HKB12] A. Heinecke, M. Klemm, and H.-J. Bungartz, "From GPGPU to Many-Core: Nvidia Fermi and Intel Many Integrated Core Architecture," *Computing in Science & Engineering*, vol. 14, no. 2, pp. 78–83, Mar. 2012. 2.4.1

[HP00] I. Hiskens and M. Pai, "Trajectory sensitivity analysis of hybrid systems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 2, pp. 204–220, 2000. 1.2.4

[HP02] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, 3rd Edition*. Morgan Kaufmann, 2002. 2.1

[HR88] M. Haque and A. Rahim, "An efficient method of identifying coherent generators using Taylor series expansion," *IEEE Transactions on Power Systems*, vol. 3, pp. 1112–1118, 1988. 3.3.1

[HSL14] HSL, "A collection of Fortran codes for large scale scientific computation." 2014. [Online]. Available: http://www.hsl.rl.ac.uk/ 4.9, 5.8

[HSS08] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of 7th Python in Science Conference (SciPy2008)*, vol. 836, Pasadena, 2008, pp. 11–15. 6

[HSV81] G. Hachtel and A. Sangiovanni-Vincentelli, "A survey of third-generation simulation techniques," *Proceedings of the IEEE*, vol. 69, pp. 1264–1280, 1981. 4.6

[HW96] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*, 2nd ed., ser. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. 1.2.2.1

[HWYC09] L. Huwang, Y. H. T. Wang, A. B. Yeh, and Z. S. J. Chen, "On the exponentially weighted moving variance," *Naval Research Logistics*, vol. 56, pp. 659–668, 2009. 4.6.3.2

[ILM98] F. Iavernaro, M. La Scala, and F. Mazzia, "Boundary values methods for time-domain simulation of power system dynamic behavior," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, no. 1, pp. 50–63, 1998. 3.3.3

[IS90] M. Irving and M. Sterling, "Optimal network tearing using simulated annealing," *IEE Proceedings C (Generation, Transmission and Distribution)*, vol. 137, no. 1, pp. 69–72, 1990. 3.3.1

[ISCP87] M. Ilic'-Spong, M. L. Crow, and M. A. Pai, "Transient Stability Simulation by Waveform Relaxation Methods," *IEEE Transactions on Power Systems*, vol. 2, no. 4, pp. 943–949, 1987. 3.3.3

[Ish08] A. Ishchenko, "Dynamics and stability of distribution networks with dispersed generation," Ph.D. dissertation, Eindhoven University of Technology, 2008. 1.3.1

[JMD09] V. Jalili-Marandi and V. Dinavahi, "Instantaneous Relaxation-Based Real-Time Transient Stability Simulation," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1327–1336, Aug. 2009. 2.4.2, 3.3.1, 3.3.3

[JMD10] ——, "SIMD-Based Large-Scale Transient Stability Simulation on the Graphics Processing Unit," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1589–1599, Aug. 2010. 2.4.1, 3.3.2

[JMZD12] 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, Jul. 2012. 2.4.1, 3.3.3

[JW01] Y.-L. Jiang and O. Wing, "A note on convergence conditions of waveform relaxation algorithms for nonlinear differential-algebraic equations," *Applied Numerical Mathematics*, vol. 36, no. 2-3, pp. 281–297, Feb. 2001. 3.3.3

[KB00] B. Kim and R. Baldick, "A comparison of distributed optimal power flow algorithms," *IEEE Transactions on Power Systems*, vol. 15, no. 2, pp. 599–604, May 2000. 2.4.2

[KD13] H. Karimipour and V. Dinavahi, "Accelerated parallel WLS state estimation for large-scale power systems on GPU," in *Proceedings of 2013 North American Power Symposium (NAPS)*, Manhatan, Sep. 2013. 2.4.1

[Kel95] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995. A.1

[KM09] S. K. Khaitan and J. McCalley, "Fast parallelized algorithms for on-line extended-term dynamic cascading analysis," in *Proceedings of 2009 IEEE PES Power Systems Conference and Exposition (PSCE)*, Seattle, Mar. 2009. 2.4.2

[KPG92] A. Kulkarni, M. Pai, and S. Ghoshal, "Parallel computation of power system dynamics using multistep methods," *International Journal of Electrical Power & Energy Systems*, vol. 14, no. 1, pp. 33–38, Feb. 1992. 2.4.2

[KRF92] D. P. Koester, S. Ranka, and G. C. Fox, "Power Systems Transient Stability-A Grand Computing Challenge," NPAC, Tech. Rep. August, 1992. 3.3.1, 5.1

[KRF94] ——, "A parallel Gauss-Seidel algorithm for sparse power system matrices," in *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, New York, 1994. 2.4.2

[Kro63] G. Kron, *Diakoptics: the piecewise solution of large-scale systems*. London: MacDonald, 1963. 3.3.3

[KSY+91] J. Kubokawa, H. Sasaki, N. Yorino, N. Okubo, J. Takehara, and M. Kitagawa, "A parallel computation of state estimation by transputer," in *Proceedings of 1991 International Conference on Advances in Power System Control, Operation and Management*, Hong Kong, 1991. 2.4.2

[Kun94] P. Kundur, *Power system stability and control*. McGraw-hill New York, 1994. 1.2

[LB93] M. La Scala and A. Bose, "Relaxation/Newton methods for concurrent time step solution of differential-algebraic equations in power system dynamic simulations," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 5, pp. 317–330, May 1993. 1.2.2, 2.4.2

[LBTC90] M. La Scala, A. Bose, D. Tylavsky, and J. Chai, "A highly parallel method for transient stability analysis," *IEEE Transactions on Power Systems*, vol. 5, no. 4, pp. 1439–1446, Nov. 1990. 3.3.3

[LBTT90] M. La Scala, M. Brucoli, F. Torelli, and M. Trovato, "A Gauss-Jacobi-Block-Newton method for parallel transient stability analysis (of power systems)," *IEEE Transactions on Power Systems*, vol. 5, no. 4, pp. 1168–1177, 1990. 2.4.2, 3.3.3

[LDG+12] P. Li, C. Ding, F. Gao, H. Yu, X. Guo, Y. Zhou, and C. Wang, "The parallel algorithm of transient simulation for distributed generation powered micro-grid," in *Proceedings of 2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, Tianjin, May 2012. 2.4.3

[LDTY11] Z. Li, V. D. Donde, J.-C. Tournier, and F. Yang, "On limitations of traditional multi-core and potential of many-core processing architectures for sparse linear solvers used in large-scale power system applications," in *Proceedings of 2011 IEEE PES General Meeting*, Detroit, Jul. 2011. 2.4.1, 2.4.3

[LJ15] Y. Liu and Q. Jiang, "Two-Stage Parallel Waveform Relaxation Method for Large-Scale Power System Transient Stability Simulation," *IEEE Transactions on Power Systems*, pp. 1–10, 2015. 3.3.3

[LL14] X. Li and F. Li, "GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method," *Electric Power Systems Research*, vol. 116, pp. 87–93, Nov. 2014. 2.4.1

[LSS94] M. La Scala, G. Sblendorio, and R. Sbrizzai, "Parallel-in-time implementation of transient stability simulations on a transputer network," *IEEE Transactions on Power Systems*, vol. 9, no. 2, pp. 1117–1125, May 1994. 2.4.2, 3.3.3

[LST91] M. La Scala, R. Sbrizzai, and F. Torelli, "A pipelined-in-time parallel algorithm for transient stability analysis (power systems)," *IEEE Transactions on Power Systems*, vol. 6, no. 2, pp. 715–722, May 1991. 2.4.2, 3.3.3

[LT95] C. Lemaitre and B. Thomas, "Two applications of parallel processing in power system computation," in *Proceedings of 1995 Power Industry Computer Applications Conference*, Salt Lake City, 1995, pp. 62–69. 2.4.2

[MBB08] J. Machowski, J. Bialek, and J. Bumby, *Power system dynamics: stability and control*. JohnWiley & Sons, 2008. 1.2, 1.2.2, 1.2.2.1, 1.2.2.1, 1.2.5, 1.2.5.1, 1.2.5.2, 1.2.5.2, 4.4, 5.3.4

[McK04] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st conference on computing frontiers on Computing frontiers*, New York, 2004, p. 162. 2.1

[Mil10] F. Milano, *Power System Modelling and Scripting*, ser. Power Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. 1.2.2.1, 1.2.5.1, 1.2.5.2, 1.2.5.2, A.1

[MMR10] B. Marinescu, B. Mallem, and L. Rouco, "Large-scale power system dynamic equivalents based on standard and border synchrony," *IEEE Transactions on Power Systems*, vol. 25, no. 4, pp. 1873–1882, 2010. 5.1

[Mor99] B. Morini, "Convergence behaviour of inexact Newton methods," *Mathematics of Computation*, vol. 68, no. 228, pp. 1605–1614, Mar. 1999. 4.7

[MQ92] N. Muller and V. H. Quintana, "A sparse eigenvalue-based approach for partitioning power networks," *IEEE Transactions on Power Systems*, vol. 7, pp. 520–527, 1992. 3.3.1

[MRR12] M. McCool, J. Reinders, and A. Robison, *Structured parallel programming: patterns for efficient computation*. Elsevier/Morgan Kaufmann, 2012. 2.1, 2.4.3, 2.5, 2.5.1, 2.5.3, 2.5.4, 2.6.1, 2.6.2, 2.6.2, 2.6.3, 2.8

[MSM04] T. G. Mattson, B. A. Sanders, and B. L. Massingill, *Patterns for Parallel Programming*. Addison Wesley Software Patterns Series, 2004. 2.1, 2.3.1, 2.3.2

[Mul01] U. A. Muller, "Specially weighted moving averages with repeated application of the ema operator," Olsen and Associates, Zurich, Switzerland, Tech. Rep., 2001. 4.6.3.2, 4.6.3.2, 4.6.3.2

[NAA11] K. M. Nor and M. Abdel-Akher, "Parallel three-phase load flow analysis for large scale unbalanced distribution networks," in *Proceedings of 2011 International Conference on Power Engineering, Energy and Electrical Drives*, Malaga, May 2011. 2.4.3

[Nie64] J. Nievergelt, "Parallel methods for integrating ordinary differential equations," *Communications of the ACM*, vol. 7, pp. 731–733, 1964. 3.3.3

[NMT+06] J. Nieplocha, A. Marquez, V. Tipparaju, D. Chavarria-Miranda, R. Guttromson, and H. Huang, "Towards efficient power system state estimators on shared memory computers," in *Proceedings of 2006 IEEE PES General Meeting*, Montreal, 2006. 2.4.3

[OKS90] T. Oyama, T. Kitahara, and Y. Serizawa, "Parallel processing for power system analysis using band matrix," *IEEE Transactions on Power Systems*, vol. 5, no. 3, pp. 1010–1016, 1990. 2.4.2

[OO96] Y. Oda and T. Oyama, "Fast calculation using parallel processing and pipeline processing in power system analysis," *Electrical Engineering in Japan*, vol. 116, no. 5, pp. 85–96, 1996. 2.4.2

[PAGV14] F. Plumier, P. Aristidou, C. Geuzaine, and T. Van Cutsem, "A relaxation scheme to combine Phasor-Mode and Electromagnetic Transients Simulations," in *Proceedings of 18th Power System Computational Conference (PSCC)*, Warsaw, 2014. C.5

[PLGMH11] F. Pruvost, P. Laurent-Gengoux, F. Magoules, and B. Haut, "Accelerated Waveform Relaxation methods for power systems," in *Proceedings of 2011 International Conference on Electrical and Control Engineering*, Wuhan, 2011, pp. 2877–2880. 4, 3.3.3

[Pod78] R. Podmore, “Identification of Coherent Generators for Dynamic Equivalents,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-97, 1978. 3.3.1

[Prz63] J. S. Przemieniecki, “Matrix Structural Analysis of Substructures,” *AIAA Journal*, vol. 1, pp. 138–147, 1963. 3.1

[QH13] Z. Qin and Y. Hou, “A GPU-Based Transient Stability Simulation Using Runge-Kutta Integration Algorithm,” *International Journal of Smart Grid and Clean Energy*, vol. 2, no. 1, pp. 32–39, 2013. 2.4.1

[RM09] J. Rommes and N. Martins, “Exploiting structure in large-scale electrical circuit and power system problems,” *Linear Algebra and its Applications*, vol. 431, no. 3-4, pp. 318–333, Jul. 2009. 6.2, 6.2

[RR13] T. Rauber and G. Rünger, *Parallel programming: For multicore and cluster systems*. Springer-Verlag Berlin Heidelberg, 2013. 2.2.3

[RR14] L. Rakai and W. Rosehart, “GPU-Accelerated Solutions to Optimal Power Flow Problems,” in *Proceedings of 47th Hawaii International Conference on System Sciences*, Hawaii, Jan. 2014, pp. 2511–2516. 2.4.1

[RSI85] M. Rafian, M. Sterling, and M. Irving, “Decomposed load-flow algorithm suitable for parallel processor implementation,” *IEE Proceedings C Generation, Transmission and Distribution*, vol. 132, no. 6, p. 281, 1985. 2.4.2

[SA10] J. Singh and I. Aruni, “Accelerating Power Flow studies on Graphics Processing Unit,” in *Proceedings of 2010 Annual IEEE India Conference (INDICON)*, Kolkata, Dec. 2010. 2.4.1

[Saa03] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed. Society for Industrial and Applied Mathematics, 2003. 3.1, 3.2.1, 3.2.3.1, 3.2.3.1, 3.2.3.2, 5.4

[Sch08] J. Schlabbach, “Low voltage fault ride through criteria for grid connection of wind turbine generators,” in *Proceedings of 5th International Conference on the European Electricity Market*, Lisboa, May 2008. 5.7, 5.8.1

[SCM98] E. Solodovnik, G. Cokkinides, and A. S. Meliopoulos, “On stability of implicit numerical methods in nonlinear dynamical systems simulation,” in *Proceedings of 13th Southeastern Symposium on System Theory*, 1998, pp. 27–31. 1.2.2.1

[SCM08] G. Stefopoulos, G. Cokkinides, and A. Meliopoulos, “Quadratic integration method for transient simulation and harmonic analysis,” in *Proceedings of 13th International Conference on Harmonics and Quality of Power*, Wollongong, 2008. 1.2.2.1

[SL85] A. Saleh and M. Laughton, “Cluster analysis of power-system networks for array processing solutions,” *IEE Proceedings C Generation, Transmission and Distribution*, vol. 132, no. 4, p. 172, 1985. 2.4.2

[SS99] Y. Saad and M. Sosonkina, “Distributed Schur complement techniques for general sparse linear systems,” *SIAM Journal on Scientific Computing*, 1999. 3.2.3.1

[Sto79] B. Stott, “Power system dynamic response calculations,” *Proceedings of the IEEE*, vol. 67, no. 2, pp. 219–241, 1979. 1.2.3, 1.2.5

[SVCC77] A. Sangiovanni-Vincentelli, L.-K. Chen, and L. Chua, “An efficient heuristic cluster algorithm for tearing large-scale networks,” *IEEE Transactions on Circuits and Systems*, vol. 24, 1977. 3.3.1

[SW03] M. Shahidehpour and Y. Wang, *Communication and Control in Electric Power Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Jun. 2003. 2.4.2

[SXZ05] J. Shu, W. Xue, and W. Zheng, “A Parallel Transient Stability Simulation for Power Systems,” *IEEE Transactions on Power Systems*, vol. 20, no. 4, pp. 1709–1717, Nov. 2005. 2.4.2, 3.3.2

[TAT81] H. Taoka, S. Abe, and S. Takeda, “Multiprocessor system for power system analysis,” *Annual Review in Automatic Programming*, vol. 11, pp. 101–106, Jan. 1981. 2.4.2

[TC95] M. Ten Bruggencate and S. Chalasani, “Parallel implementations of the power system transient stability problem on clusters of workstations,” in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, San Diego, 1995, p. 34. 2.4.2, 3.3.2

[TDL11] J.-C. Tournier, V. Donde, and Z. Li, “Potential of General Purpose Graphic Processing Unit for Energy Management System,” in *Proceedings of 6th International Symposium on Parallel Computing in Electrical Engineering*, Luton, Apr. 2011, pp. 50–55. 2.4.1

[TM72] C. Tavora and O. M. Smith, “Characterization of Equilibrium and Stability in Power Systems,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-91, no. 3, pp. 1127–1130, May 1972. 1.2.6

[TOG14] S. Tabik, G. Ortega, and E. M. Garzón, “Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study,” *The Journal of Supercomputing*, vol. 70, no. 2, pp. 577–587, Jan. 2014. 2.4.1

[TW67] W. Tinney and J. Walker, “Direct solutions of sparse network equations by optimally ordered triangular factorization,” *Proceedings of the IEEE*, vol. 55, no. 11, pp. 1801–1809, 1967. A.1

[TW05] A. Toselli and O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*, ser. Springer Series in Computational Mathematics. Berlin-Heidelberg: Springer-Verlag, 2005, vol. 34. 3.1, 3.2.1, 4

[UBP10] F. M. Uriarte and K. L. Butler-Purry, “Multicore simulation of an AC-radial Shipboard Power System,” in *Proceedings of 2010 IEEE PES General Meeting*, Minneapolis, Jul. 2010. 2.4.3

[UD13] F. M. Uriarte and C. Dufour, “Multicore methods to accelerate ship power system simulations,” in *Proceedings of 2013 IEEE Electric Ship Technologies Symposium (ESTS)*, Arlington, Apr. 2013, pp. 139–146. 2.4.3

[UH11] F. M. Uriarte and R. Hebner, “Development of a multicore power system simulator for ship systems,” in *Proceedings of 2011 IEEE Electric Ship Technologies Symposium*, Alexandria, Apr. 2011, pp. 106–110. 2.4.3

[VCB92] G. Vuong, R. Chahine, and S. Behling, “Supercomputing for power system analysis,” *IEEE Computer Applications in Power*, vol. 5, no. 3, pp. 45–49, Jul. 1992. 2.4.2

[VFK92] M. Vale, D. Falcao, and E. Kaszkurewicz, “Electrical power network decomposition for parallel computations,” in *Proceedings of 1992 IEEE International Symposium on Circuits and Systems*, vol. 6, San Diego, 1992. 3.3.1

[VGL06] T. Van Cutsem, M. E. Grenier, and D. Lefebvre, “Combined detailed and quasi steady-state time simulations for large-disturbance analysis,” *International Journal of Electrical Power and Energy Systems*, vol. 28, pp. 634–642, 2006. C.3.1

[VLER12] F. Villella, S. Leclerc, I. Erlich, and S. Rapoport, “PEGASE pan-European testbeds for testing of algorithms on very large scale power systems,” in *Proceedings of 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Berlin, Oct. 2012. 1.3.3

[VMMO11] C. Vilacha, J. C. Moreira, E. Miguez, and A. F. Otero, “Massive Jacobi power flow based on SIMD-processor,” in *Proceedings of 10th International Conference on Environment and Electrical Engineering*, Rome, May 2011. 2.4.1

[VP13] T. Van Cutsem and L. Papangelis, “Description, Modeling and Simulation Results of a Test System for Voltage Stability Analysis,” University of Liege, Tech. Rep. November, 2013. [Online]. Available: http://hdl.handle.net/2268/141234 1.3.1, 4.9.1, 4.9.1.1

[VV13] G. Valverde and T. Van Cutsem, "Control of dispersed generation to regulate distribution and support transmission voltages," in *Proceedings of IEEE PES 2013 PowerTech conference*, Grenoble, Jun. 2013. 1.3.1, 5.8.2

[WC76] Y. Wallach and V. Conrad, "Parallel solutions of load flow problems," *Archiv für Elektrotechnik*, vol. 57, no. 6, pp. 345–354, Nov. 1976. 2.4.2

[WEC14] "Wind Power Plant Dynamic Modeling Guide," Western Electricity Coordinating Council (WECC), Tech. Rep., 2014. 1.3.1

[WHS99] F. Wang, N. Hadjsaid, and J. Sabonnadière, "Power system parallel computation by a transputer network," *Electric Power Systems Research*, vol. 52, no. 1, pp. 1–7, Oct. 1999. 2.4.2

[Woh01] B. I. Wohlmuth, *Discretization Methods and Iterative Solvers Based on Domain Decomposition*, ser. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, vol. 17. 3.1, 4

[XCCG10] Y. Xu, Y. Chen, L. Chen, and Y. Gong, "Parallel real-time simulation of Integrated Power System with multi-phase machines based on component partitioning," in *Proceedings of 2010 International Conference on Power System Technology*, Zhejiang, Oct. 2010. 2.4.3

[Yan14] R. R. O. Yang, "A Comparison of EMT, Dynamic Phasor, and Traditional Transient Stability Models," M.Sc. Thesis, University of Manitoba, 2014. C.5

[YRA93] S. B. Yusof, G. J. Rogers, and R. T. H. Alden, "Slow coherency based network partitioning including load buses," *IEEE Transactions on Power Systems*, vol. 8, pp. 1375–1381, 1993. 3.3.1

[YXZJ02] L. Yalou, Z. Xiaoxin, W. Zhongxi, and G. Jian, "Parallel algorithms for transient stability simulation on PC cluster," in *Proceedings of 2002 International Conference on Power System Technology*, vol. 3, no. 1, Kunming, 2002, pp. 1592–1596. 2.4.2, 3.3.2

[ZCC96] X.-P. Zhang, W.-J. Chu, and H. Chen, "Decoupled asymmetrical three-phase load flow study by parallel processing," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 143, no. 1, p. 61, 1996. 2.4.2

[ZD14] 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, Jun. 2014. 2.4.1