Parallel Computation of Large Power System Networks Using the Multi-Area Thévenin Equivalents

by

Marcelo Aroca Tomim

B.Sc., Escola Federal de Engenharia de Itajubá, 2001
M.A.Sc., Universidade Federal de Itajubá, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Electrical & Computer Engineering)

The University Of British Columbia

(Vancouver)

July, 2009

Abstract

The requirements of today's power systems are much different from the ones of the systems of the past. Among others, energy market deregulation, proliferation of independent power producers, unusual power transfers, increased complexity and sensitivity of the equipments demand from power systems operators and planners a thorough understanding of the dynamic behaviour of such systems in order to ensure a stable and reliable energy supply.

In this context, on-line Dynamic Security Assessment (DSA) plays a fundamental role in helping operators to predict the security level of future operating conditions that the system may undergo. Amongst the tools that compound DSA is the Transient Stability Assessment (TSA) tools, which aim at determining the dynamic stability margins of present and future operating conditions.

The systems employed in on-line TSA, however, are very much simplified versions of the actual systems, due to the time-consuming transient stability simulations that are still at the heart of TSA applications. Therefore, there is an increasing need for improved TSA software, which has the capability of simulating bigger and more complex systems in a shorter lapse of time.

In order to achieve such a goal, a reformulation of the *Multi-Area Thévenin Equivalents* (MATE) algorithm is proposed. The intent of such an algorithm is parallelizing the solution of the large sparse linear systems associated with transient stability simulations and, therefore, speeding up the overall on-line TSA cycle. As part of the developed work, the matrix-based MATE algorithm was re-evaluated from an electric network standpoint, which yielded the network-based MATE presently introduced. In addition, a performance model of the proposed algorithm is developed, from which a theoretical speedup limit of $\frac{p}{2}$ was deduced, where p is the number of subsystems into which a system is torn apart. Applications of the network-based MATE algorithm onto solving actual power systems (about 2,000 and 15,000 buses) showed the attained speedup to closely follow the predictions made with the formulated performance model, even on a commodity cluster built out of inexpensive out-of-the-shelf computers.

Table of Contents

List of Tables

List of Figures

Glossary

Acronyms

BBDF	Block-Bordered Diagonal Form
BLAS	Basic Linear Algebra Subroutines
CG	Conjugate Gradient
CGS	Conjugate Gradient Squared
COTS	Comommodity (or Commercial) Off-The-Shelf
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access With Collision Detection
DSA	Dynamic Security Assessment
FACTS	Flexible Alternating Current Transmission System
flop	floating-point operation
GPU	Graphical Processing Unit
HVDC	High-Voltage Direct-Current transmission
IP	Internet Protocol
LAPACK	Linear Algebra PACKage
MATE	Multi-Area Thévenin Equivalents
Mflops	10^6 flops per second
NIC	Network Interface Card
PC	Personal Computer
PDE	Partial Differential Equations

RAM	Random-access memory
RMS	Root Mean Squared
SCI	Scalable Coherent Interface
SCTP	Stream Control Transmission Protocol
SMP	Symmetric Multiprocessing
SSBI	South-Southeastern Brazilian Interconnected system
SVC	Static-Var Compensator
TCP	Transmission Control Protocol
TSA	Transient Stability Assessment
VDHN	Very DisHonest Newton
WECC	Western Electricity Coordinating Council system

Mathematical Terms

$\mu(\cdot)$	Arithmetic mean
$\sigma(\cdot)$	Standard deviation
$\mathcal{O}(\cdot)$	Indicate the same order of compexity of the function it operates upon
$(\cdot)^{-1}$	Inverse matrix operator
$(\cdot)^{T}$	Tranpose matrix operator
$\Im\{\cdot\}$	imaginary part
$\Re\{\cdot\}$	real part
j	pure imaginary number, i.e., $\sqrt{-1}$
$(\cdot)^{*}$	complex comjugate operator
$\bar{E}$,$\bar{V}$,$\bar{I}$,$\bar{Z}$	RMS phasor quantities
ρ	The ratio of branches to buses in $\mathbb{S}$

b_k	Number of border nodes contained in $\mathbb{B}_k$
k	Subsystem index, where $k = 1, \ldots, p$
l	Number of global links contained in $\mathbb{L}$
l_k	Number of local links contained in $\mathbb{L}_k$
n	Number of nodes in the untorn system $\mathbb{S}$
N_i	Number of times the current injections associated with the untorn system $\mathbb{S}$ change
n_k	Number of internal nodes contained in $\mathbb{N}_k$
N_z	Number of times the untorn system $\mathbb{S}$ undergoes topological changes
p	Number of subsystems, which the system $\mathbb{S}$ is torn into
$\mathbb{B}_k$	Set of the b_k border nodes, which are connected to the subsystem $\mathbb{S}_k \in \mathbb{S}$
$\mathbb{L}_k$	Set of the l_k local links, which are connected to the subsystem $\mathbb{S}_k \in \mathbb{S}$
$\mathbb{L}$	Set of the l global links, which interconnect all subsystems $\mathbb{S}_k \in \mathbb{S}$
$\mathbb{N}_k$	Set of the n_k internal nodes, which belong to the subsystem $\mathbb{S}_k \in \mathbb{S}$
$\mathbb{S}_k$	k^{th} subsystem associated with $\mathbb{S}$, i.e., $\mathbb{S}_k \in \mathbb{S}$
$\mathbb{S}$	Original untorn system
$\mathbf{e}_k^b$	Thévenin voltages vector at the b_k border nodes of $\mathbb{S}_k$, i.e., in $\mathbb{B}_k$
$\mathbf{e}_k^l$	Vector of dimension l with the Thévenin voltages of the subsystem $\mathbb{S}_k$, i.e., in $\mathbb{B}_k$, which are connected to the global links in $\mathbb{L}$
$\mathbf{i}_k^b$	Vector of dimension b_k with current injections into the border nodes in $\mathbb{B}_k$
$\mathbf{v}_k^b$	Vector of dimension b_k with voltage drops across subsystem $\mathbb{S}_k$, or seen from the border nodes in $\mathbb{B}_k$
$\mathbf{v}_k^l$	Vector of dimension l with voltage drops across subsystem $\mathbb{S}_k$, or seen $\mathbb{B}_k$, due to the global link currents in $\mathbb{L}$
$\mathbf{e}_k$	Thévenin voltages vector at all n_k nodes in $\mathbb{N}_k$

$\mathbf{i}_k$	Current injections vector of dimension n_k due to the local links in $\mathbb{L}_k$
$\mathbf{j}_k$	Internal current injections vector of dimension n_k associated with the system $\mathbb{S}_k$
$\mathbf{v}_k$	Nodal voltages vector of dimension n_k associated with the system $\mathbb{S}_k$
$\mathbf{i}$	Current injections vector of dimension n associated with the system $\mathbb{S}$
$\mathbf{v}$	Nodal voltages vector of dimension n associated with the system $\mathbb{S}$
$\mathbf{e}^l$	Multi-area Thévenin voltages vector of dimension l exciting the global links $\mathbb{L}$
$\mathbf{i}^l$	Global link currents vector of dimension l associated with the global links in $\mathbb{L}$
$\mathbf{Z}_k^b$	Multi-node Thévenin impedance matrix of dimension $b_k \times b_k$ with respect to the border nodes in $\mathbb{B}_k$
$\mathbf{Z}_0^l$	Primitive impedance matrix of dimension $l \times l$ associated with the global links in $\mathbb{L}$
$\mathbf{Z}_k^l$	Link Thévenin impedance matrix of dimension $l \times l$ associated with the subsystem $\mathbb{S}_k$
$\mathbf{Z}^l$	Multi-area Thévenin impedance matrix of dimension $l \times l$ associated with the global links $\mathbb{L}$
$\mathbf{L}_k$	Lower diagonal factor of the admittance matrix $\mathbf{Y}_k$, obtained from the LU factorization
$\mathbf{P}_k$	Link-to-subsystem transformation matrix of dimension $n_k \times l$, which maps $\mathbb{L}$ onto $\mathbb{N}_k$. The subsystem-to-link transformation matrix is given by $(\mathbf{P}_k)^T$
$\mathbf{Q}_k$	Subsystem-to-border transformation matrix of dimension $b_k \times n_k$, which maps $\mathbb{N}_k$ onto $\mathbb{B}_k$. The border-to-subsystem transformation matrix is given by $(\mathbf{Q}_k)^T$
$\mathbf{R}_k$	Link-to-border transformation matrix of dimension $b_k \times l$, which maps $\mathbb{L}$ onto $\mathbb{B}_k$. The border-to-link transformation matrix is given by $(\mathbf{R}_k)^T$
$\mathbf{U}_k$	Upper diagonal factor of the admittance matrix $\mathbf{Y}_k$, obtained from the LU factorization
$\mathbf{Y}_k$	Admittance matrix of dimension $n_k \times n_k$ associated with each subsystem $\mathbb{S}_k$
$\mathbf{Y}$	Admittance matrix of dimension $n \times n$ associated with the system $\mathbb{S}$

$\mathcal{E}_{MATE}$	Efficiency achieved by the parallel network-based MATE algorithm when solving the system $\mathbb{S}$
$\mathcal{S}_{MATE}$	Speedup achieved by the parallel network-based MATE algorithm over the sequential sparse linear solver when solving the system $\mathbb{S}$
$g(m)$	Minimum time, or gap, between consecutive message transmissions or receptions, which also considers all contributing factors, including $o_s(m)$ and $o_r(m)$
L	End-to-end latency from process to process, which includes the time for copying the data to and from the network interfaces and transferring the data over the physical network
$o_r(m)$	Period of time in which the processor is engaged in receiving the message of size m
$o_s(m)$	Period of time in which the processor is engaged in sending the message of size m
T_C	Timing associated with the communication overhead incurred by the network-based MATE algorithm
T_L	Timing associated with the sequential tasks performed by the process handling the link solve
T_{MATE}	Time spent by the parallel network-based MATE solver to solve the original system $\mathbb{S}$
T_P	Timing associated with parallel tasks performed by the processes handling subsystems
T_{SPARSE}	Time spent by the sequential sparse linear solver to solve the original system $\mathbb{S}$
$T_{1fact}(\mathbb{S}_k)$	Time spent performing the first-time factorization of $\mathbf{Y}_k$.
$T_{2fact}(\mathbb{S}_k)$	Time spent performing the same-pattern factorization of $\mathbf{Y}_k$. In this case, the symbolic factorization, required in the first-time factorization, is not repeated
$T^i_{comm}(\mathbb{L})$	Time spent scattering the border node current injections $\mathbf{i}^b_k$ to the subsystems $\mathbb{S}_k \in \mathbb{S}$
$T^i_{comm}(\mathbb{S}_k)$	Time spent receiving the border nodes injections $\mathbf{i}^b_k$ from the link solver

$T_{comm}^{Thv}(\mathbb{L})$	Time spent receiving $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$ from the subsystems $\mathbb{S}_k \in \mathbb{S}$
$T_{comm}^{Thv}(\mathbb{S}_k)$	Time spent sending $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$ to the link solver process
$T_{comp}^{i}(\mathbb{L})$	Time spent setting up and computing the link current equations
$T_{comp}^{Thv}(\mathbb{S}_k)$	Time spent computing the multi-node Thévenin equivalent $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$ for the subsystem $\mathbb{S}_k \in \mathbb{S}$
$T_{comp}^{v}(\mathbb{S}_k)$	Time spent computing the nodal voltages in the subsystem $\mathbb{S}_k$
$T_{comp}^{v}(\mathbb{S}_k)$	Time spent solving the node voltages $\mathbf{v}_k$ for the updated current injections $\mathbf{i}_k$
$T_{fact}(\mathbb{L})$	Time spent performing the dense factorization of $\mathbf{Z}^l$.
$T_{idle}^{lnk}(\mathbb{S}_k)$	Time spent waiting for the link solver process to compute the link currents $\mathbf{i}^l$
$T_{idle}^{subs}(\mathbb{L})$	Time spent waiting for the subsystems $\mathbb{S}_k \in \mathbb{S}$ to compute $\mathbf{Z}_k^b$, $\mathbf{e}_k^b$ and $\mathbf{v}_k$
$T_{solv}(\mathbb{L})$	Time spent performing the forward/backward substitutions using $\mathbf{L}_l$ and $\mathbf{U}_l$ factors associated with $\mathbf{Z}^l$
$T_{solv}(\mathbb{S}_k)$	Time spent performing the forward/backward substitutions using $\mathbf{L}_k$ and $\mathbf{U}_k$ factors associated with $\mathbf{Y}_k$
$\mathrm{T}(\mathbb{L})$	Time spent by the link solver process solving the link system $\mathbb{L}$
$\mathrm{T}(\mathbb{S}_k)$	Time spent by the processes solving the subsystem $\mathbb{S}_k$

Acknowledgements

This thesis arose, in part, out of years of research carried out by the University of British Columbia Power Engineering Group, led by Dr. José R. Martí. It is indeed a great pleasure to be part of one of the most prominent research groups on dynamic simulations of electric power systems.

In particular, I would like to thank Dr. José R. Martí for the supervision, advice and guidance throughout the Ph.D. program, which was vital to the conduct of the present work.

Since I started the Ph.D. program, I had the opportunity to know a great number of people of the most varied cultural and professional backgrounds, which contributed in assorted ways to my growth as a person and a researcher. To those, I would like to record my most sincere acknowledgements.

A special mention should be made to Mazana Armstrong for introducing me to the previous works related to my thesis topic, to Michael Wrinch for his valuable comments and advice in both scientific and professional realm and, to Tom De Rybel for introducing me to the Linux world and his ceaselessly technical support with the computing cluster.

I would also like to thank the CAPES Foundation (Fundação de Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) of Brazil for funding my Ph.D. program. In addition, I would like to thank the British Columbia Transmission Corporation and the Powertech Labs Inc. for extending the funding of this project and providing the data relative to the Western Electricity Coordinating Council system, which was used in the performance analysis presented in this work. I would also like to acknowledge Dr. Lei Wang of Powertech Labs Inc. for the his crucial comments, which further motivated me to accomplish this work.

I gratefully thank Dr. Paulo Nepomuceno Garcia of the Federal University of Juiz de Fora (UFJF), Brazil, for further appreciating the results of this work and helping me organizing the body the thesis.

I would also to acknowledge and thank my research committee members, Dr. K. D. Srivastava, Dr. Hermann Dommel and Dr. Juri Jatskevich for their so much valuable time put into reading my thesis and for their equally important feedback that definitely helped improving the content of this thesis.

Last, but far from least, I would like to thank my family for the unconditional support, patience and love that motivates me to be a better person every day.

Dedication

To Adriana and Nicholas,
You are my life.

Chapter 1

Introduction

Modern electric power systems are often less secure than the systems of the past. This is a result of reduced attention to the transmission infrastructure caused by deregulation, proliferation of independent power producers, unusual power transfers driven by market activities, the use of complex controls and special protection schemes, and a general lack of systemwide oversight regarding reliable planning and operation (Wang & Morison, 2006).

In this context, possible types and combinations of energy transactions occurring at any given time may grow enormously. Therefore, today's power systems can no longer be operated in a structured manner (Kundur et al., 2000). One solution to mitigate this uncertainty is prediction of future operating conditions, for example through use of on-line Dynamic Security Assessment (DSA). Such a tool takes a snapshot of the system operating condition, performs a comprehensive security assessment in near-real-time, and provides the operators with warnings of abnormal situations as well as remedial measure recommendations (Wang & Morison, 2006).

One of the most costly computation-wise tools of on-line DSA is undoubtedly the Transient Security Assessment (TSA). Many alternatives to tackle this problem have been proposed in the literature (Xue et al., 1993; Mansour et al., 1995; Marceau & Soumare, 1999; Ernst et al., 2001; Kassabalidis, 2002; Juarez T. et al., 2007), which includes more efficient time domain solutions, direct stability, pattern recognition and expert systems/neural networks, as well as hybrid methods. Many of these methods, however, still rely on transient stability time-domain simulation tools, due to the high level of accuracy and flexibility of these tools in terms of complexity of the system models. Hence, speeding up TSA tools can significantly improve overall performance of on-line DSA.

In the past two decades, dramatic improvements in microprocessor technology have been observed. According to (Grama et al., 2003), the average number of cycles per instruction of high end processors has improved by roughly an order of magnitude during the 1990's. However, not only the speed of the light and the effectiveness of heat dissipation techniques impose physical limits on the speed of a single computer, but also the cost of advanced single-processor computers, which increases more rapidly than their power. As personal computer (PC) performance has increased and prices have fallen steeply, for both PCs and networks employed to interconnect them, dedicated clusters of PC workstations have become

an interesting alternative to traditional supercomputers (Gropp et al., 1999, 2003). Such a picture tends to shift high-performance computing even further towards clusters of PCs and parallelism as multi-core processors have recently become the standard.

Many have been the attempts to parallelize transient stability simulations, but only a few have aimed at taking full advantage of commodity PC clusters. Therefore, further research on porting industrial-grade transient stability simulators onto today's parallel computing systems with minimal programming effort and maximum returns in terms of computational throughput is still of significant importance. Furthermore, inexpensive parallel computing systems present themselves as a strong alternative to provide electric utilities operating centres with the so much needed real-time DSA applications.

In the sequence, further reasons that motivate parallelism and some useful performance metrics applied to analyzing parallel programs will be presented. Afterwards, a literature review on parallel algorithms applied to solving the transient stability problem will be presented. Lastly, the motivation of the present work will be introduced, followed by the major achievements resulting from the present research project.

1.1 Motivating Parallel Computing

A brief survey of trends in applications, computer architecture, and networking, presented by Foster (1995), suggests a future in which parallelism plays a vital role not only for supercomputers development but also workstations, personal computers, and networks. In this future, programs will be required to exploit the multiple processors located inside each computer and the additional processors available across a network. Moreover, because most existing algorithms are specialized for a single processor, this situation implies a need for new algorithms and program structures to be able to perform many operations at once.

As pointed out by Grama et al. (2003), the role of concurrency in accelerating computing elements has been recognized for several decades. In the past, however, when trends in hardware development were not clear and standardized parallel programming interfaces were not yet established, development of parallel software had traditionally been thought of as highly time and effort intensive. Nowadays, many are the arguments that support parallelism, such as the computational power, memory and disk speed and data communication requirements of today's applications.

On the computational power argument, it is recognized that increased computational performance will not come from faster processors, but parallel ones. This fact can be explained by the fact that processors clock cycle times[1] are decreasing in a much slower pace,

[1]The time to perform a basic operation is ultimately limited by the clock cycle.

as physical limits such as speed of light are approached. Therefore, the only way to increase the amount of operations that can be performed by a computer architecture is by executing them in parallel (Foster, 1995).

In conjunction with the speed of the processor, the latency and bandwidth of the memory system represents another factor that influences the computational performance. Usually, memory access times are higher than the execution of floating-point operations, which impose a tremendous computational performance bottleneck. The usage of hierarchical and faster memory devices, called caches, reduces the problem, but not entirely. Parallel platforms typically yield better memory system performance because they provide larger aggregate caches and higher aggregate bandwidth to the memory system (Grama et al., 2003).

As the problems increase in size and complexity, larger databases are required, which may be infeasible to collect in a central location. In such situations, parallel computing appears as the only solution. In addition, an extra motivation for computational parallelism comes in cases where a central data storage is also undesirable for unusual technical and/or security reasons.

1.1.1 Commodity Off-The-Shelf Computer Systems

Past decades' advances in computer technology made general-purpose personal computers (PCs) an alternative cost effective solution to the traditional supercomputers for computing large problems quickly.

In early 1990's, Dr. Thomas Sterling and Dr. Donald Becker, two engineers in NASA, hypothesized that using inexpensive, commodity off-the-shelf (COTS) computer systems hooked together with high-speed networking (even with speeds as low as 10 Mbit/s Ethernet) could duplicate the power of supercomputers, particularly applications that could be converted into highly parallelized threads of execution. They theorized that the price/performance of these COTS systems would more than make up for the overhead of having to send data between the different nodes to have that additional computing done (Gropp et al., 2003). Eventually, this concept became known as *Beowulf clusters* (Beowulf.org, 2009), whose generic structure is illustrated in Figure 1.1.

Another advantage of the COTS systems over the early supercomputers is the ease of programming and maintaining. Supercomputers were often hand-made systems that required specific programming techniques compliant with only a certain models. In the case of commodity clusters, common processor's architectures are employed, which allow code reuse and, therefore, significantly reduces the development and deployment time of parallel applications. Regarding the network interconnects, many types are available in the market

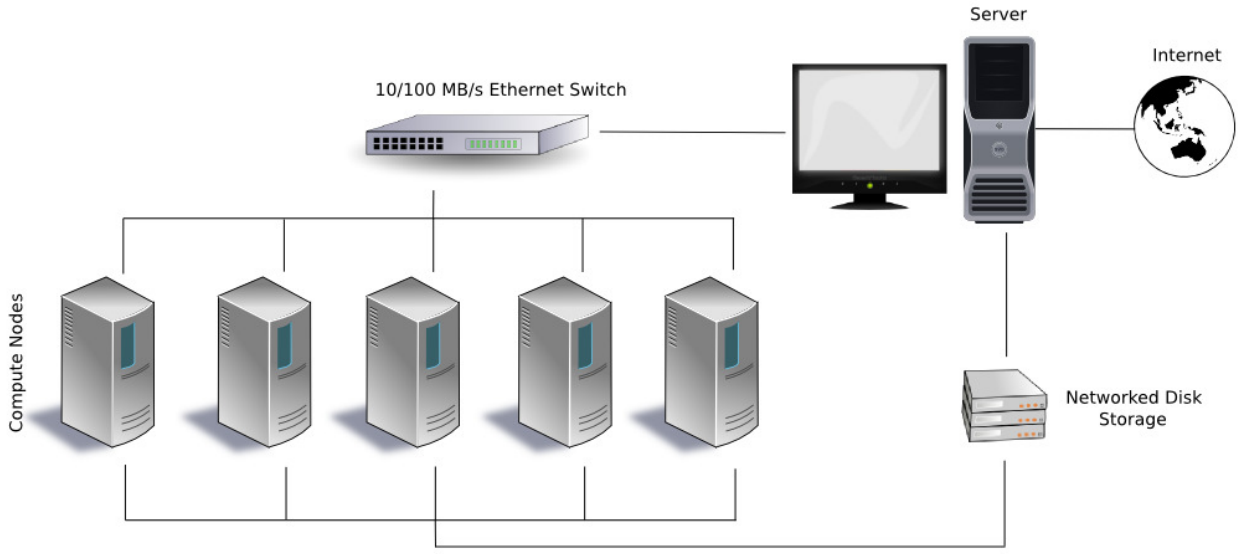


Figure 1.1. Generic COTS computer system (`http://upload.wikimedia.org/wikipedia/commons/4/40/Beowulf.png`).

with different underlying hardware approaches. However, parallel programming standards, such as *Message-Passage Interface* (MPI) and *Portable-Virtual Machine* (PVM), helped abstract programming from the hardware, which further leveraged the success of commodity computing clusters.

Nowadays, low-cost commodity clusters built from individual PCs and interconnected by private low-latency and high-bandwidth networks provide users with an unprecedented price/performance and configuration flexibility for high-performance parallel computing.

1.1.2 Performance Metrics for Parallel Systems

When analyzing the performance of parallel programs in order to find the best algorithm, a number of metrics have been used, such as speedup and efficiency.

Speedup

Speedup is a relative measure that extracts the acceleration delivered by a parallel algorithm with respect to the equivalent best known sequential algorithm. Mathematically, speedup, given in (1.1), corresponds to the ratio of the time taken to solve a specific problem using a single processor, T_s, to the time demanded to solve the same problem on a parallel

environment with p identical processors, T_p.

$$\mathcal{S} = \frac{T_s}{T_p} \tag{1.1}$$

Moreover, the parallel timing T_p can be further split into three smaller components. Even though parallel programs aim at utilizing the p available processors as much as possible, there are always parts of the problems solutions that remain sequential, which in turn incur in a sequential timing, T_{ps}. Also included in the total parallel timing T_p is the overhead timing T_{po}, which comprises extra computation timings and interprocessor communication timings. Last but not least, the actual time spent executing parallel work T_{pw}.

$$\mathcal{S} = \frac{T_s}{T_{ps} + T_{pw} + T_{po}} = \frac{1}{f_{ps} + f_{pw} + f_{po}} \tag{1.2}$$

where $f_{pk} = \frac{T_{pk}}{T_s}$ with $k \in \{s, w, o\}$ correspond their associated normalized quantities with respect to the best sequential timing T_s.

In fact, these normalized quantities are usually functions that depend on the problem to be solved, the number of available processors p and characteristics intrinsic to the hardware/software setup. Therefore, in order to find such relationships thorough understanding of the underlying parallel algorithm and the costs involved in solving a given problem in a parallel architecture is required.

For example, assume that a problem of size N can be solved in parallel by any number of processors, which can be, hypothetically, infinity. Consider also that the parallel overhead f_{po} is kept negligible for any number of processors, and the parallel work fraction f_{pw} diminishes asymptotically with the number of processors, i.e., f_{pw} tends to zero as p heads towards infinity. In such a case, one can observe that the speedup achieved with the parallel solution will never exceed $\frac{1}{f_{ps}}$. Such behavior of parallel programs was firstly observed by Amdahl (1967), who stated that the sequential portion of the program alone would place an upper limit on the speedup, even if the sequential processing were done in a separate processor.

In an ideal parallel system, however, both sequential processing required by the parallel algorithm and the overhead times are null, whereas the parallel work can approximated by the original sequential time T_s split evenly among p processors. In such a case, the time required to solve the problem in parallel would be $\frac{T_s}{p}$, which incur in a p-fold speedup. In practice, speedups greater than p (also know as *superlinear speedups*) can be observed when the work performed by the sequential program is greater than its parallel counterpart. Such phenomenon is commonly observed when data necessary to solve a problem is too large to fit into the cache of a single processor, but suitable to fit into several units, which can be

accessed concurrently (large cache aggregate).

Efficiency

Efficiency is a measure of the fraction of the time for which a processing unit is usefully employed (Grama et al., 2003). Its mathematical definition is given by the ratio of the speedup to the number of processing units used, as given below.

$$\mathcal{E} = \frac{\mathcal{S}}{p} = \frac{T_s}{p\,T_p} \tag{1.3}$$

From (1.3), it can be readily verified that the efficiency of an ideal parallel system equals the unity. In practice, however, communication overhead tend to increase as the number of processors grows, which yields efficiencies always lower than the unity.

1.2 Parallel Transient Stability Solution

Strategies for parallelizing traditional sequential transient stability solutions rely on the structure of the discretized differential-algebraic equations given below:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{v}) \tag{1.4a}$$

$$\mathbf{Y}\,\mathbf{v} = \mathbf{i}(\mathbf{x}, \mathbf{v}) \tag{1.4b}$$

where, $\mathbf{x}$ represents a vector with dynamic variables (or, state variables), whose first derivatives $\dot{\mathbf{x}}$ are defined by a vector function $\mathbf{f}$, normally dependent on $\mathbf{x}$ and the vector with nodal voltages $\mathbf{v}$. In addition, $\mathbf{i}$ represents a vector function that defines the nodal current injections, which also depend on the variable states $\mathbf{x}$ and the nodal voltages $\mathbf{v}$. Lastly, $\mathbf{Y}$ represents the complex-valued nodal admittance matrix of the system under study.

In general, differential equations (1.4a) are usually associated with dynamic devices connected to the passive network. Usually, there is no direct connection between dynamic devices and all interactions occur through the network. As a consequence, state variables are usually grouped by device, which keep them independent from the others and, hence, make them suitable for concurrent computations. As for the network equations (1.4b), parallelization is not as straightforward as for the differential equations. Generally, electric networks are highly sparse with irregular connectivity patterns, fact that causes parallel solutions for such networks very difficult to find (Tylavsky et al., 1992).

Parallelization of these two major tasks for the solution of a single time step is often referred to as *parallel-in-space* approach, because of the usage of the mathematical structure

of the problem and the system topology. Other strategies adopt *parallel-in-time* approach, where multiple time-steps of the same transient stability simulation are solved simultaneously. In practice, parallel-in-time and parallel-in-space methods are combined in order to enhance the efficiency of the overall simulation.

Various parallel algorithms for transient stability computation have been proposed but few have been actually tested on parallel machines. Moreover, many are the possible algorithmic combinations reported in the literature that take advantage of parallelization in both space and time. In the sequence, a few of these parallelization techniques applied to the transient stability problem will be quickly reviewed.

1.2.1 Parallel Newton Methods

The first parallel-in-space variation of the Newton-Raphson method[2] was introduced by Hatcher et al. (1977). The approach adopted in that work was distributing sets of discretized differential equations (1.4a) among different processors, and solving the network equations (1.4b) by means of two main algorithms: a sequential sparse LU factorization and a parallel Successive Over Relaxation (SOR) method.

Later on, Alvarado (1979) proposed a parallel-in-time approach based on the Newton-Raphson method applied to differential equations discretized by the trapezoidal integration method. Network equations, however, were not considered at that time.

La Scala et al. (1990b, 1991) formalized and extended the mathematical formulation of the previous method, yielding a parallel-in-space-and-time method. The basic idea in this technique was assigning $T \times p$ processors to solve multiple time steps, where each group of p processors computes one of the T steps of a predefined time window. More specifically, blocked Gauss-Jacobi (La Scala et al., 1990b) and Gauss-Seidel (La Scala et al., 1991) methods were proposed for the iterations between time steps, while each time step was solved by the parallel-in-space *Very DisHonest Newton* (VDHN) method. Again, network equations were computed sequentially.

Chai et al. (1991) describes the first reported implementations of a parallel transient stability simulator on two different computing architectures, a distributed and a shared-memory system. In this study the parallel VDHN method and the SOR-Newton methods were also compared. For the tested 662-bus and 91-generator system, a SOR-Newton method presented a speedup of 7.4 on the shared-memory machine and 3.5 on the distributed-memory system, when employed 8 processors. Such metrics make evident that communication-intensive methods, such as the SOR-Newton, are more efficient when implemented on shared-memory

[2]For details, see Appendix B.

systems. As for the VDHN method, speedups of 3.9 and 4.6 were reported for the same 8 processors on the shared and distributed-memory systems, respectively. It shows that, on a shared-memory system, the performance of the VDHN method is much lower than the one observed for the SOR-Newton method, due to the sequential solution of the network equations. On a distributed-memory system, however, the VDHN method performance is slightly better than SOR-Newton due to the reduced communication overhead, and it is not higher only because of the sequential solution of the network equations. In addition, the parallel-in-space VDHN method was used to compute multiple time steps simultaneously, yielding a parallel-in-space-and-time VDHN method. When solving the same 662-bus system employing 32 processors on the distributed-memory system, the performance of this combined method achieved a speedup of about 9 times in comparison to the sequential VDHN method. Later, Chai & Bose (1993) summarize practical experiences with parallel Gauss-type, VDHN, SOR-Newton, Maclaurin-Newton and Newton-W matrix methods, where the later one is discussed in Section 1.3.

Parallel implementations of the VDHN method on a research-purpose and a production-grade transient stability simulators are presented in (Wu et al., 1995). The adopted approach was a step-by-step solution of the entire system, with differential equations and network equations split among several processors on a shared-memory system (Cray). The parallel network solutions was based on the methodology introduced in (Huang & Wing, 1979; Wu & Bose, 1995). As part of the results of the paper, parallelization of the differential equations are shown to be practically linearly scalable with number of processors in such architecture (about 16 times speedup with 20 processors, only for the machine equations computations). As for the parallel network equations factorization and triangular solutions, a strong speedup saturation is observed, which achieved speedups about 11 times for the factorization and 5 times for triangular solutions on 20 processors. The overall speedup of the parallel transient stability solutions was about 7 times on the same 20 processors.

Decker et al. (1996) introduces a new class of parallel iterative solvers into the Newton-based transient stability solutions, Conjugate Gradient (CG) based methods. Both parallel-in-time and space approaches were considered in solving the transient stability problem on a distributed-memory architecture. Moreover, network equations were also parallelized as part of the CG-like iterative solutions. Despite of high level of parallelism provided by such methods, intensive communication overhead along with convergence issues considerably degraded the performance of the algorithm with respect to sequential solutions.

Other parallel implementations of the VDHN method, which employs same parallel sparse factorization and triangular solution suggested by Huang & Wing (1979), are discussed in (La Scala et al., 1996; Hong & Shen, 2000). According to (La Scala et al., 1996), the parallel

VDHN method implemented on a shared-memory system was able to achieve roughly 7.7 times speedup on 20 processors, when solving a system with 662 buses and 91 generators. On an 8-processor distributed-memory system, Hong & Shen (2000) reports speedups of about 3.6 and 5.5 times when solving systems with 1017 buses (150 generators) and 3021 buses (450 generators), respectively.

Hong & Shen (2000) also apply the Gauss-Seidel method, proposed in (La Scala et al., 1991), on a distributed-memory architecture. Taking the sequential VDHN method as a reference, the parallel-in-time-and-space Gauss-Seidel method achieved 9 and 13 times speedup when solving the aforementioned 1017-bus and 3021-bus systems, respectively, on 32 processors.

1.2.2 Parallel Waveform Relaxation Methods

Waveform relaxation methods (WRM) had been shown to be very effective for the transient analysis of Very-Large-Scale-Integration (VLSI) circuits and was introduced to transient stability analysis of large power systems by Ilić-Spong et al. (1987). The basic idea of the WRM is to solve the waveform of one state variable, considering all other state variables waveforms fixed at their previous values. The same is then repeated for the other state variables using the updated waveforms until convergence is observed. This method is similar to Gauss-Jacobi method, but applied to the whole state variable waveform. Although predictions were made with respect to the linear scalability of the method when implemented on a truly parallel computing environment, no comparisons with the best available transient stability solution, based on Newton-like algorithms, were presented. Improvements on the parallel WRM applied to the transient stability analysis are proposed in (Crow & Ilić, 1990), which included aspects related to windowing and partitioning of the state variables among distinct processors. Results showed that the method is linearly scalable, although the presented simulations did not considered communication overhead.

In another attempt in solving the transient stability problem by means of WRM-like methods, La Scala et al. (1994, 1996) proposed the Shifted-Picard method, also known as the WRM-Newton method for parallelizing the transient stability problem. As synthesized in the article, the main idea of the algorithm consists in linearizing nonlinear differential-algebraic equations (DAEs) about a given initial guess waveform. Then, the guess waveform is updated by solving a linear set of DAEs derived by linearization. The original equations should be then re-linearized about the updated guess in an iterative fashion. Experiments with a system with 662 buses and 91 generators on an distributed system, which took a sequential VDHN-based simulator as the reference, show that this algorithm achieved about 7 times

speedup with 32 processors, which represents an efficiency of 22% per processor. For a system with 2583 buses and 511 generators, Aloisio et al. (1997) showed that performance Shifted-Picard method is drastically decreased due to intensive interprocessors communication on a distribute-memory architecture (IBM SP2). In such a case, the observed speedup with respect to the serial VDHN algorithm was 0.5 on 8 processors. Moreover, network equations were solved sequentially.

Wang (1998) proposed a parallel-in-time relaxed Newton method, as an improvement to the SOR-Newton method. An implementation of the proposed method on a distributed-memory Cray-T3D presented an speedup of about 9.4 times at 47% efficiency on 20 processors, with respect to a sequential VDHN-based transient stability program.

1.2.3 Parallel Alternating Methods

As observed in Section 1.2, in the alternating solution approach the differential equations can be straightforwardly solved concurrently in a parallel-in-space fashion. The network equations, however, still remain as a major problem to be efficiently parallelized, due to its irregular connectivity pattern.

In order to overcome the sequentiality of the network equations solutions, La Scala et al. (1990a) exploited the in-space parallelism inherent to the SOR method, at the expense of the quadratic convergence proper of the Newton-based methods, for the network solutions give in (4.4b).

Another solution strategy, which relies on the network decomposition, originally proposed in (Ogbuobiri et al., 1970; Huang & Wing, 1979), was used by Decker et al. (1992, 1996). In this work, the network equations were solved iteratively by a parallel implementation of the Conjugate Gradient (CG) method on a distributed multiprocessor machine. In spite of the high level of parallelism inherent to the CG method, speedup of up to 7 were observed when 32 processors were employed in the computations.

Also based on network decomposition (Ogbuobiri et al., 1970; Huang & Wing, 1979), Shu et al. (2005) adopted a node-tearing technique (discussed in Section 1.3 approach for parallelizing the network equations on a cluster of multi-core processors.

1.3 Parallel Network Solutions

The vast majority of problems encountered in engineering demands the solution of large sparse linear systems, expressed mathematically by (1.5).

$$\mathbf{Ax} = \mathbf{b} \tag{1.5}$$

where $\mathbf{x}$ represents the the vector of unknowns, $\mathbf{b}$ is known vector, usually specifying boundary or contour conditions, and the $\mathbf{A}$ is a large problem-dependent sparse square matrix.

In power systems engineering, more specifically, such linear systems are often associated with large electric networks which can span entire continents. As such networks present irregular and sparse connectivity patterns, their associated nodal equations (represented by the matrix $\mathbf{A}$ in (1.5)) end up assuming the same topological pattern. Irregular sparsity is due to the fact that in large power systems nodes are directly connected to just a few other neighboring nodes in a variety of ways (Sato & Tinney, 1963). Since Tinney & Walker (1967), sparsity techniques applied to the solution of large sparse linear systems of equations have been heavily studied. As a consequence, direct sparse LU factorization became the standard tool for solving sparse linear systems, which arise from many power systems problems, such as fault analysis, power flow analysis and transient stability simulations.

Although sparsity techniques have been extremely successful on sequential computers, parallel algorithms that take full advantage of the sparsity of the power systems problems are more difficult to develop. The difficulty is mainly due to the aforementioned irregularity of power networks, in addition to the lack of evident parallelism in the required computational operations. In order to address the problem, a number of parallel algorithms have been proposed in the literature, which can be categorized into two major classes, fine and coarse grain parallelization schemes. Independently of the class, these algorithms often need a preprocessing stage that aims at properly scheduling independent tasks on different processors. Topology of the networks and their factorization paths described by means of graphs provide extremely useful information which help exploit the inherent parallelism in the computations, even though these are not readily evident.

1.3.1 Fine Grain Schemes

In fine grain schemes, parallelism is achieved by dividing the factorization process and forward/backward substitutions into a sequence of small elementary tasks and scheduling them on several processors.

In (Huang & Wing, 1979; Wing & Huang, 1980; Wu & Bose, 1995), these elementary

tasks consist of one or two floating-point operations, which are scheduled based on precedence relationships defined by the system's topology and node ordering. An illustration of the method is given in Figure 1.2. In this example, the list of required operations to eliminate the sparse matrix[3] is given, along with its associated task graph. The main idea is to take advantage of the independence of the tasks belonging to a same level of the task graph and performing them in parallel. An optimal scheduling based on the levels of the task graph associated with the optimally-ordered system matrix was suggested in (Huang & Wing, 1979). Wu & Bose (1995) extend the previous work and presents an implementation of the algorithm on a shared-memory computer. For the parallel LU factorization, results show almost linearly-scalable speedups of up to 17 times on 20 processors, i.e., an efficiency of 85%, while for the parallel forward/backward substitutions, speedups strongly saturate at 8 processors and peak at about 5.5 times when 16 processors are used.

Employing a different approach, in (Alvarado et al., 1990; Enns et al., 1990), the available parallelism was exploited from the $\mathbf{W}$ matrix defined below.

$$\mathbf{W} = \mathbf{L}^{-1} = (\mathbf{L}_1 \, \mathbf{L}_2 \, \ldots \, \mathbf{L}_m)^{-1} = \mathbf{L}_m^{-1} \, \ldots \, \mathbf{L}_2^{-1} \, \mathbf{L}_1^{-1}$$

where,

$$\mathbf{A} = \mathbf{L} \, \mathbf{D} \, \mathbf{L}^T$$

and $\mathbf{D}$ is a diagonal matrix and $\mathbf{L}$ is a lower diagonal factor matrix of $\mathbf{A}$.

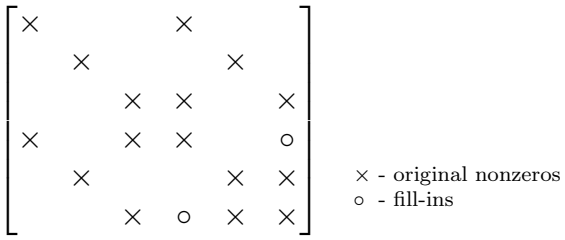
Based on the fact that each matrix $\mathbf{L}_k^{-1}$ for $k = 1, \ldots, m$ describes one update operation[4] required in the Gaussian elimination, parallelism can be obtained by properly aggregating sets of $\mathbf{L}_k^{-1}$, so $\mathbf{W} = \mathbf{W}_a \, \mathbf{W}_b \, \ldots \, \mathbf{W}_z$. Each intermediate multiplication is then performed in parallel. Indirect performance measurements of this methodology, when applied to the transient stability problem on a shared-memory machine, are discussed in (Chai & Bose, 1993). A maximum speedup of about 5 times was observed on 16 processors, and it strongly saturated for higher number of processors.

1.3.2 Coarse Grain Schemes

Schemes of this nature rely on the fact that electric power systems can be partitioned into smaller subsystems, which are in turn interconnected or interfaced by a limited number of branches or nodes. As long as all subsystems are independent from one another, apart from the interconnection or interface system, each subsystem can be solved concurrently

[3] Notice, this is a *perfect elimination matrix* defined in Section A.3.

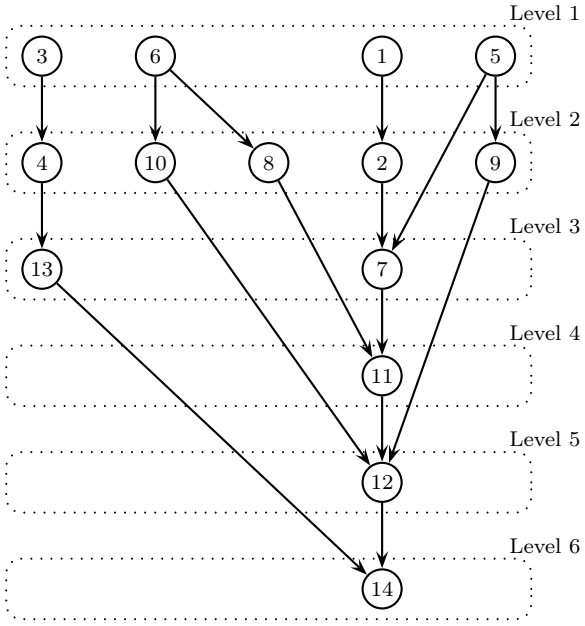
[4] For further details on the LU factorization, see Appendix A.



(a) Sparse matrix

Task	Operation
1	$a_{14} \leftarrow {}^{a_{14}}/_{a_{11}}$
2	$a_{44} \leftarrow a_{44} - a_{41}a_{14}$
3	$a_{25} \leftarrow {}^{a_{25}}/_{a_{22}}$
4	$a_{55} \leftarrow a_{55} - a_{52}a_{25}$
5	$a_{34} \leftarrow {}^{a_{34}}/_{a_{33}}$
6	$a_{36} \leftarrow {}^{a_{36}}/_{a_{33}}$
7	$a_{44} \leftarrow a_{44} - a_{43}a_{34}$
8	$a_{46} \leftarrow a_{46} - a_{43}a_{36}$
9	$a_{64} \leftarrow a_{64} - a_{63}a_{34}$
10	$a_{66} \leftarrow a_{66} - a_{63}a_{36}$
11	$a_{46} \leftarrow {}^{a_{46}}/_{a_{44}}$
12	$a_{66} \leftarrow a_{66} - a_{64}a_{46}$
13	$a_{56} \leftarrow {}^{a_{56}}/_{a_{55}}$
14	$a_{66} \leftarrow a_{66} - a_{65}a_{56}$

(b) Task list



(c) Task graph

Figure 1.2. Sparse matrix and its associated task graph.

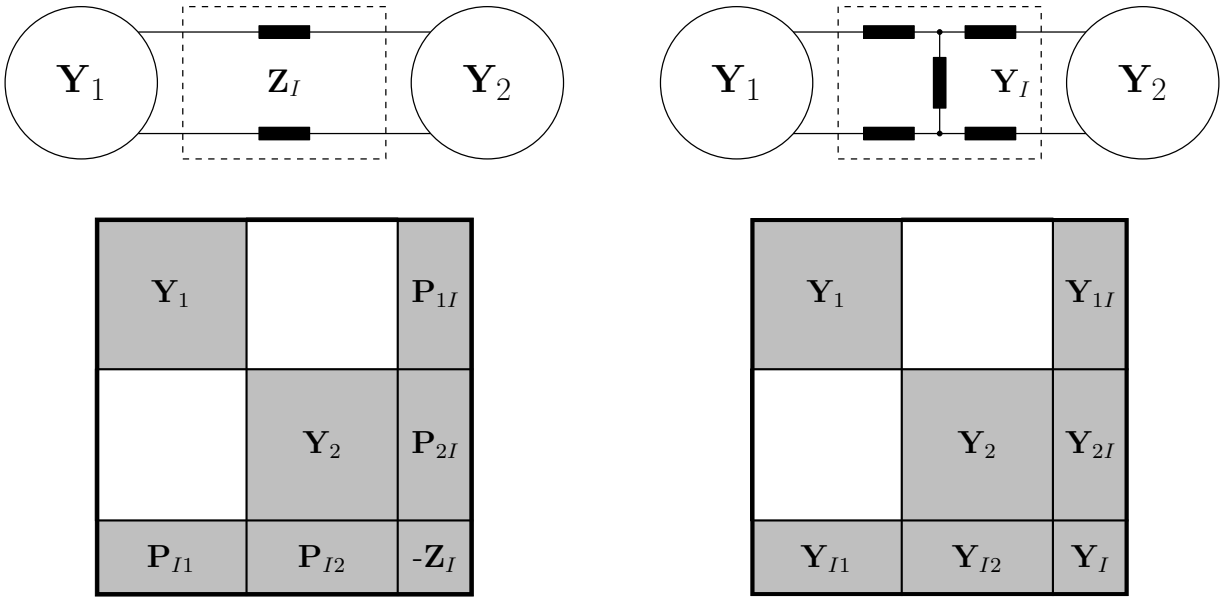
on distinct processors. Therefore, partitioning algorithms of large systems into independent subsystems is at the heart of any coarse grain parallel scheme.

The basic partitioning schemes for electrical networks are based on branch and node tearing algorithms, illustrated in Figure 1.3.

Branch tearing techniques

In branch tearing methods, subsystems are completely disconnected from each other when the interconnecting branches are removed from the system. This group of branches is also know as the *cut-set* of the system, and is represented by the currents associated with the interconnection system $\mathbf{Z}_I$ in Figure 1.3a.

The most well-known example of such technique is *Diakoptics*, developed by Gabriel Kron and his followers (Happ, 1970, 1973; Kron, 1953, 1963). As pointed out by Happ (1973, 1974),



(a) Branch tearing. **(b)** Node tearing.

Figure 1.3. Partitioning techniques.

the basic idea of diakoptics is to solve a large system by breaking, or tearing, it apart into smaller subsystems; to first solve the individual parts, and then to combine and modify the solutions of the torn parts to yield the solution of the original untorn problem. The combination or modification of the torn solutions so that they apply to the untorn original problem is the crux of the method. It was precisely this task that Kron accomplished through a series of contour transformations and link divisions.

Many authors attempted to clarify the concepts proposed by Kron and further developed by Happ. Among them, Wu (1976) explained the basic idea of diakoptics as merely the partitioning of the branches and the Kirchhoff's laws. He also observed that practical large networks are usually sparsely connected, and thus sparse matrix techniques, pioneered by Tinney & Walker (1967), could be used in making diakoptics more computationally efficient. Alvarado et al. (1977) also investigated the feasibility of employing diakoptics as a means of solving large networks, and concluded that sparsity techniques are fundamental in improving the performance of the diakoptics-based algorithms proposed at that time.

More recently, the Multi-Area Thévenin Equivalents (MATE) method was proposed by Martí et al. (2002), which extends the idea of diakoptics in terms of interconnected Thévenin equivalents. As summarized by the authors, the MATE algorithm embodies the concepts of multi-node Thévenin equivalents, diakoptics, and the modified nodal analysis by Ho et al. (1975), in a simple and general formulation. Another application of the MATE algorithm is demonstrated by Hollman & Martí (2003), who employed a distributed-memory

PC-cluster for calculating electromagnetic transients in real-time. Although the work by Hollman & Martí (2003) has its foundations on the MATE algorithm, it further exploits computational parallelism from the time decoupling provided by transmission lines existent in power systems (Dommel, 1996). This technique completely eliminates the need for combining the subsystems' solutions in order to obtain the solution of the untorn system and, therefore, only requires exchanges of past values of voltages (i.e., known values) between subsystems interconnected by a given transmission line. Timings for the solution of a 234-node/349- branch system show an achieved speedup of about 3.5 times using 5 PCs.

Before the present work, no parallel implementation of any branch tearing based algorithm applied to the solution of large power systems has been found in the literature. Performance predictions of proposed algorithms, however, were presented in (Wu, 1976; Alvarado et al., 1977).

Node tearing techniques

In node tearing methods, subsystems are completely disconnected from each other when the nodes that lie in a common interface ($\mathbf{Y}_I$) are removed from the system, as depicted in Figure 1.3b. Node tearing or network decomposition was firstly proposed as a means to reorder the sparse matrices in order to reduce fill-ins during the LU factorization, as suggested by Tinney & Walker (1967) and Ogbuobiri et al. (1970). Due to the matrix shape illustrated in Figure 1.3b, network decomposition is also often referred to as the Block-Bordered Diagonal Form (BBDF) method.

At first, parallel applications based on such partitioning techniques were only qualitatively evaluated. Brasch et al. (1979) showed that BBDF-based parallel algorithms would severely suffer from scalability issues, due to the strong speedup saturation predicted for increasing number of processors. Lau et al. (1991) presented a partitioning algorithm based on the computation of precedence relationships embedded in the factorization paths described by Tinney et al. (1985). The partitioned factorization path was then structured according to BBDF and properly scheduled on distinct processors of a distributed-memory system. Results confirmed previous predictions, where speedups strongly saturated for more than 4 processors. For a 662-bus system, the maximum observed speedups were slightly inferior to 2 times on the same 4 processors, which leads to an efficiency of 50%. Other implementations of parallel BBDF methods can be found in the literature, although the timings are not clearly separated from the global timings of the problem solution, such as the transient stability problem, discussed previously.

1.4 Other Related Work

Parallel computing has been employed in solving scientific and engineering problems for decades. Structural analysis of materials, weather forecast, simulation of astronomical phenomena, simulation of air flow about an aircraft and calculation of electromagnetic fields around electric equipments are just a few examples of problems that are extensively tackled by parallel computing due to their computation and data intensiveness.

A computational task shared by all the previous problems is the solution of large sparse linear systems of equations, often extracted from sets of discretized ordinary or partial differential equations. Many parallel algorithms have been proposed to solve such large sparse systems. These can be categorized into parallel direct methods and parallel iterative methods.

1.4.1 Parallel Direct Methods

Parallel direct solvers are based on the classical Gaussian elimination combined with partial or total pivoting strategies. Amongst the most well known parallel direct sparse solvers available are the SuperLU DIST (Li & Demmel, 2002) and MUMPS (Amestoy & Duff, 2000). The algorithms underlying these two solvers represent a vast class of solvers (Amestoy et al., 2001). Hence, an overview of such solvers, to some extent, summarizes two of the best algorithms employed in computer science for direct sparse linear solutions.

SuperLU DIST partitions the matrix in a two dimensional block-cyclic fashion. The blocks are formed by groups of submatrices and assigned to a two-dimensional process grid of shape $r \times c$, as depicted in Figure 1.4. Each block is defined based on the notion of *unsymmetric supernodes*, suggested by Demmel et al. (1999). As described in (Li & Demmel, 2002), a supernode is a range of columns of $\mathbf{L}$ with the triangular block just below the diagonal being full, and the same nonzero structure elsewhere (either full or zero). The off-diagonal block may be rectangular and need not be full. By the block-cyclic layout, the block (i, j) is mapped onto the process at coordinate $\big((i-1) \mod r, (j-1) \mod c\big)$ of the process grid. During the factorization, the block $\mathbf{L}(i, j)$ is only needed by the process row $\big((i-1) \mod r\big)$, which restricts the interprocess communication. Similarly, the block $\mathbf{U}(i, j)$ is only needed by the processes on the process column $\big((j-1) \mod c\big)$.

As also reported in (Li & Demmel, 2002), SuperLU DIST was able to solve matrices of various sizes and patterns, originated from different fields of application, with decreasing timings for up to 256 processors of a Cray T3E-900.

On the other hand, MUMPS (Multifrontal Massively Parallel Solver) aims at solving unsymmetric or symmetric positive definite linear systems of equations on distributed mem-

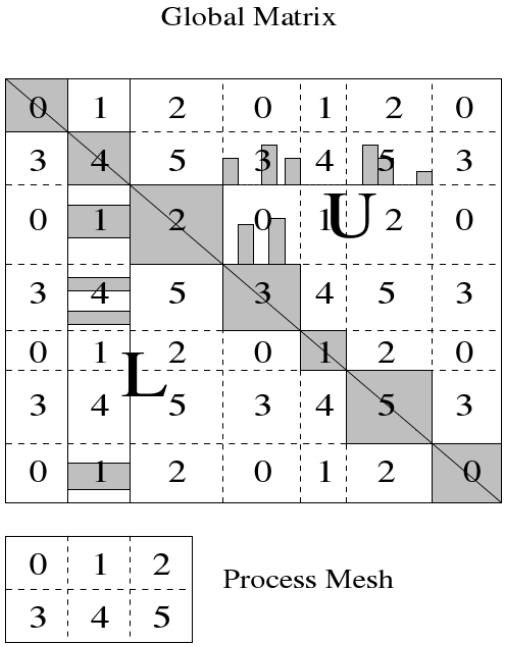


Figure 1.4. SuperLU DIST two-dimensional block-cyclic partitioning scheme. (Extracted from (Li & Demmel, 2002))

ory computing architectures. There are two main levels of parallelism within MUMPS: *tree* and *node parallelism*. As explained by Amestoy & Duff (2000), in the factorization process, data is first assembled at a node combining the Schur complements from the children nodes with data from the original matrix. The original matrix's data comprises rows and columns corresponding to variables that the analysis forecasts should be eliminated at this node. This data is usually supplied in so-called arrowhead format (i.e., BBDF format), with the matrix ordered according to the permutation from the analysis phase. This data and the contribution blocks from the children nodes are then assembled (or summed) into a frontal matrix using indirect addressing (sometimes called an extended add operation). Since the nodes belonging to a same level of the elimination tree and their children are independent (see Figure 1.5), the previous procedure can be performed concurrently for all nodes at same level. This explains the tree parallelism. The node parallelism lies on the fact that the Schur compliments of each node of the elimination tree are obtained from local blocked update operations, which can be performed by means of parallel implementations of the Level 3 BLAS (Blackford et al., 2002; Goto & Van De Geijn, 2008b).

According to Amestoy et al. (2001), although the total volume of communication is comparable for both solvers, MUMPS requires many fewer messages, especially with large numbers of processors. The difference found was up to two orders of magnitude. This is partly intrinsic to the algorithms, and partly due to the 1D (MUMPS) versus 2D (SuperLU) matrix

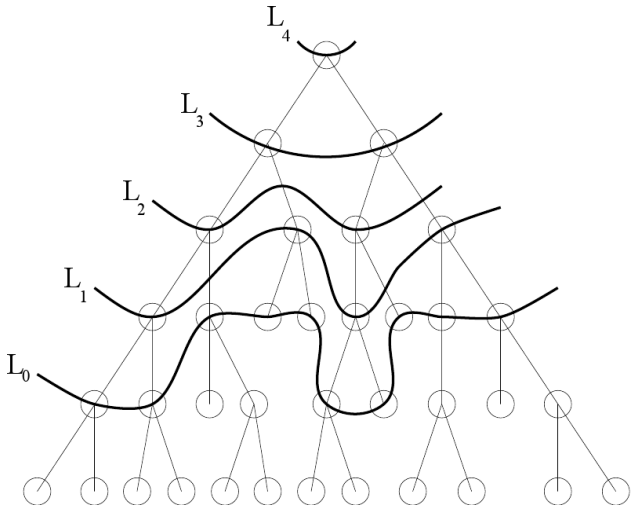


Figure 1.5. Decomposition of the elimination tree adopted within MUMPS

partitioning. Furthermore, MUMPS was, in most tested matrices, faster in both factorization and solve phases. The speed penalty for SuperLU partly comes from the code complexity that is required to preserve the irregular sparsity pattern, and partly because of the greater number of communication messages. With more processors, SuperLU showed better scalability, because its 2D partitioning scheme keeps all of the processors busier despite the fact that it introduces more messages.

In order to illustrate the performance of these tools on a commodity cluster, a series of timings were obtained as part of this thesis from the SuperLU DIST on a 16 AMD Athlon^{TM} 64 2.5 GHz processors cluster interconnected by a Dolphin SCI network is presented on Table 1.1. Both factorization and forward and backward substitutions (F/B Subst.) procedures were timed for two complex admittance matrices extracted from the South-Southeastern Brazilian Interconnected (SSBI) system (see Section 4.4) and the North American Western Electricity Coordinating Council (WECC) system (see Section 3.4). As it can be observed, regardless of the number of processors employed in the computations, the parallel timings are always greater that the timing for a single processor. This seemingly contradictory fact is explained by the additional overhead incurred by the interprocessors communications. The timings also show a marginal improvement for the WECC system (14,327 nodes) in comparison to the timings recorded for the SSBI system (1,916 nodes). According to the obtained timings, the parallel factorization is roughly 3 to 25 times slower than its sequential counterpart for the SSBI system, and 3 to 20 for the WECC system; while the parallel F/B substitutions are about 10 to 45 slower for the SSIB system, and 7 to 30 times slower for the WECC system. It can also be observed that the choice of the process grid topology also greatly influences the timings. In this case, the factorization benefits from

Table 1.1. SuperLU DIST timings on a 16 AMD Athlon^{TM} 64 2.5GHz processors cluster built on a single rack and interconnected by a dedicated network.

Process Grid	SSBI (1,916 nodes)		WECC (14,327 nodes)	
	Fact. [s]	F/B Subst. [s]	Fact. [s]	F/B Subst. [s]
1	0.0030	0.0005	0.0265	0.0047
1×2	0.0100	0.0121	0.0794	0.0916
2×1	0.0274	0.0051	0.2039	0.0427
1×3	0.0105	0.0146	0.0816	0.1063
3×1	0.0457	0.0054	0.3387	0.0404
2×2	0.0307	0.0113	0.2197	0.0829
4×1	0.0170	0.0051	0.3688	0.0383
1×4	0.0355	0.0195	0.0796	0.1419
5×1	0.0264	0.0050	0.5256	0.0348
1×5	0.0578	0.0183	0.0781	0.1305
2×3	0.0186	0.0100	0.2120	0.0714
3×2	0.0299	0.0073	0.3374	0.0516
6×1	0.0768	0.0046	0.5701	0.0316
1×6	0.0098	0.0226	0.0750	0.1591

column-wise partitioning, while row-wise partitioning benefits the F/B substitutions.

An explanation for such low performance is twofold. Firstly, algorithms as the one employed by SuperLU DIST have massive supercomputers as their target architectures, whose processors communicate by means of specialized low-latency and high-bandwidth network interconnects. Secondly, the dimension of the problems aimed at by such tools are much bigger than the power systems related problems. One can understand *bigger* as those problems for which the computational burden per processor is much *bigger* than the communication overhead due to the parallelization. In this sense, the foregoing power systems admittance matrices (∼15,000), often considered big in the power systems field, turn out to be small in comparison to problems often encountered in the computer science field, for which tools, such as MUMPS and SuperLU DIST, are developed.

1.4.2 Parallel Iterative Methods

According to Saad & Vorst (2000), originally, the usage of iterative methods was restricted to systems related to elliptic partial differential equations, discretized with finite difference

techniques[5]. For other problems, for instance those related to finite element modeling and large electric and electronic circuit calculations, direct solution techniques were preferred, because of the lack of robustness of iterative solvers for a large number of classes of matrices. Until the end of 1980's, almost none of the commercial packages for finite element problems included iterative solution techniques. However, for many PDE-related problems, the complexity of the elimination process increases too much to make realistic 3D modeling feasible. For instance, irregularly structured finite element problems of order one million may be solved by direct methods - given a large enough computer (memory wise), but at a tremendous cost and difficulty. However, some of such problems can be solved with less resources and more efficiently by means of iterative solution techniques, if an adequate preconditioning can be constructed.

As reported by Pai & Dag (1997), iterative solution techniques applied to both static and dynamic simulation problems, typical of large scale power systems, still cannot compete with direct methods because of possible convergence problem. Such statement agrees with Saad & Vorst (2000), who reported that large electric and electronic circuits are not easy to solve in an efficient and reliable manner by iterative methods.

The performance of direct methods, both for dense and sparse systems, is largely bound by the factorization of the matrix. This operation is absent in iterative methods, which also lack dense matrix suboperations. Since such operations can be executed at very high efficiency on most current computer architectures, a lower flop rate for iterative than for direct methods is expected. Furthermore, the basic operations in iterative methods often use indirect addressing, which depends on the data structure of the matrix. Such operations also have a relatively low efficiency of execution. On the other hand, iterative methods are usually simpler to implement than direct methods, and since no full factorization has to be stored, they can handle much larger systems than direct methods (Barrett et al., 1994).

Vorst & Chan (1997) compares an iterative with a direct method for a sparse system related to a finite element discretization of a second order PDE over an irregular grid, where n is the order of the problem. It is shown that the cost of a direct method varies with $n^{\frac{3}{2}}$ for a 2D grid and with $n^{\frac{5}{3}}$ for a 3D grid. Comparing then the flop counts for the direct method with those for the Conjugate Gradient (CG) method, it was concluded that the CG method might be preferable, in case different systems have to be solved each time with large n. Furthermore, in case many systems with several different right-hand sides have to be solved, it seems likely that direct methods will be more efficient, for 2D problems, as long as the number of right-hand sides is so large that the costs for constructing the LU factorization

[5]Such systems are originated from oil reservoir engineering, weather forecasting, electronic device modeling, etc.

is relatively small. For 3D system, however, it is concluded that such a choice is unlikely, because the flop counts for two triangular solves associated with a direct solution method are proportional to $n^{\frac{5}{3}}$, whereas the number of flops for the iterative solver varied according to $n^{\frac{4}{3}}$.

Following the argument presented in the previous section on parallel direct methods, it can be concluded that iterative solvers are recommended for solving even bigger problems than the ones tackled by parallel direct solvers, because of mainly memory constraints. For instance, as reported in (Li & Demmel, 2002), SuperLU DIST has played a critical role in the solution of a long-standing problem of scattering in a quantum system of three charged particles. This problem requires solving a complex, nonsymmetric and very ill-conditioned sparse linear system, whose dimension reached 8 million. The task performed by SuperLU DIST was building the block diagonal preconditioners for the Conjugate Gradient Squared (CGS) iterative solver. For a block of size 1 million, SuperLU DIST took 1209 seconds to factorize using 64 processors of a IBM SP and 26 seconds to perform triangular solutions. The total execution time of the problem was about 1 hour. This scientific breakthrough result was reported in a cover article of *Science* magazine (Rescigno et al., 1999).

1.5 Thesis Motivation

Based on the methodologies and experiences reported in the literature overviewed in the previous sections, one can readily identify the parallel solution of power networks as one of the major bottlenecks in achieving fully parallel transient stability simulations.

Although many have addressed the problem of parallelizing the solution of large sparse linear systems, no standard methodology has yet been agreed upon. This can be partially explained by the fact that some algorithms are more suitable for shared-memory systems than for distributed-memory ones, and vice-versa.

From the hardware standpoint, inexpensive commodity clusters, also known as Beowulf clusters (Gropp et al., 2003; Beowulf.org, 2009), provide a technical and economically alternative to the expensive vector-based supercomputers. Nowadays, high performance computing clusters can be easily built with off-the-shelf symmetric multiprocessing (SMP) machines interconnected through high-speed and high-bandwidth network cards. Moreover, due to limitations on increasing the number of cores on a single chip, the cluster setup is expected to become the standard in the computer industry.

Bearing the aforementioned high performance computing architectures, the most successful parallel algorithms will likely combine coarse and fine grain approaches. In this sense, by means of coarse grain approaches, one can assign tasks to SMP nodes, which in turn can

further extract further parallelism by adopting fine grain approaches.

From a software point of view, much effort had been put into developing direct sparse linear solvers. In this way, algorithms that require a complete redesign and re-implementation of existing sparse solvers are bound to be treated with a high level of scepticism mainly due to economical reasons. Such an undertaking would certainly require re-educating developers with the use of new algorithms, which in turn would require from developers time for programming and testing the new routines. The reusability of available and highly-tested sparse routines becomes significant to reduce costs and time of deployment of newly developed parallel tools.

In this context, the Multi-Area Thévenin Equivalents (MATE) approach, proposed by Martí et al. (2002) and extended by Armstrong et al. (2006), has the potential to provide both coarse and fine granularities required by SMP clusters. Moreover, the MATE approach also allows one to employ ready-to-use sparse routines, as it will be shown in this thesis. As a consequence, with a minimum programming effort, it is expected a performance boost in the solution of large sparse systems, which are at the heart of essential power system analysis tools, such as the on-line transient stability assessment.

1.6 Thesis Contributions

The main contributions of this thesis are:

- Introduction of the network-based MATE algorithm, which further optimizes the original matrix-based MATE algorithm formulation (Martí et al., 2002) in terms of computation and communication overhead;
- Implementation of the network-based MATE algorithm on a commodity cluster, built with single-core nodes interfaced by a dedicated high-speed network, employing ready-to-use sparsity libraries;
- Development of a performance model for the network-based MATE algorithm, which enabled the establishment of a theoretical speedup limit for the method with respect to traditional sequential sparsity-oriented sparse linear solvers;
- Application of the parallel network-based MATE algorithm for the solution of the network equations associated with transient stability simulations.

1.7 Publications

- Tomim, M. A., De Rybel, T., & Martí, J. R. (2009). Multi-Area Thévenin Equivalents Method Applied To Large Power Systems Parallel Computations. *IEEE Transactions on Power Systems* (submitted).

- Tomim, M. A., Martí, J. R., & Wang, L. (2009). Parallel solution of large power system networks using the Multi-Area Thévenin Equivalents (MATE) algorithm. *International Journal of Electrical Power & Energy Systems*, In Press.

- Tomim, M. A., Martí, J. R., & Wang, L. (2008). Parallel computation of large power system network solutions using the Multi-Area Thévenin Equivalents (MATE) algorithm. In *16th Power Systems Computation Conference, PSCC2008* Glasgow, Scotland.

- De Rybel, T., Tomim, M. A., Singh, A., & Martí, J. R. (2008). An introduction to open-source linear algebra tools and parallelisation for power system applications. In *Electrical Power & Energy Conference, Vancouver, Canada*

Chapter 2

Network-based Multi-Area Thévenin Equivalents (MATE)

The Multi-Area Thévenin Equivalents (MATE) method was proposed by Martí et al. (2002), which extends the idea of diakoptics in terms of interconnected Thévenin equivalents. As summarized by the authors, the MATE algorithm embodies the concepts of multi-node Thévenin equivalents, diakoptics, and the modified nodal analysis by Ho et al. (1975), in a simple and general formulation.

Although the possibility of parallel computations at the subsystems level has been conceived in (Martí et al., 2002; Armstrong et al., 2006), no implementation of the MATE algorithm in a distributed computer architecture had been realized until recently. This fact can be attributed to the only recent release of economically viable commodity clusters, built from out-of-the-self personal computers interconnected by dedicated local networks, and multi-core processors. Tomim et al. (2008, 2009) present an evaluation of the MATE algorithm applicability to the solution of nodal equations of large electric power systems in a distributed computer architecture. The MATE algorithm was implemented using ready-to-use highly optimized sparse and dense matrix routines in a 16-computer cluster interconnected by a dedicated network.

In this chapter, a formulation of the MATE algorithm from an electrical network perspective will be set forth. In comparison to the algorithm proposed by Martí et al. (2002), this network-based approach adds novel concepts to the algorithm, and further optimizes the amount of operations, memory usage, and data exchange among parallel processes during the actual solution. However, before introducing the network-based MATE algorithm, the the original problem will be restated along with the original MATE formulation. Lastly, the matrix and network-oriented approaches for formulating the MATE algorithm will be qualitatively and quantitatively compared.

2.1 Problem Statement

In order to solve an electric network, one needs to compute the voltages at all its nodes due to the excitation sources to which the system is subjected. A widely used technique for solving power systems is the well-known nodal approach (Grainger & Stevenson, 1994), where the system is modeled as an admittance matrix $\mathbf{Y}$ and all excitation sources converted into current injections, which populate the vector $\mathbf{i}$.

$$\mathbf{Y}\,\mathbf{v} = \mathbf{i} \tag{2.1}$$

Then, taking into consideration the relationship between the nodal voltages $\mathbf{v}$ and the current injections $\mathbf{i}$, given by a linear system of equations (2.1), one can finally compute $\mathbf{v}$ through Gaussian elimination, for instance.

A second approach, the one analyzed presently in greater detail, considers tearing the system apart into smaller subsystems, then solving the individual parts, and subsequently combining and modifying the solutions of the torn parts to yield the solution of the original untorn problem. In this case, assume the system $\mathbb{S}$ is subdivided into p disconnected areas, or *subsystems*, such that as $\mathbb{S} = \{\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_p\}$, and that these subsystems are interconnected by a set of *global links* $\mathbb{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_l\}$, with l elements. The nodes that form the interface between the subsystem $\mathbb{S}_k$, with $k = 1, \ldots, p$, and the interconnection system $\mathbb{L}$ will be called *border nodes* and form the set $\mathbb{B}_k$ with b_k nodes. Finally, the group of links connected to the subsystem $\mathbb{S}_k$ are called *local links* and form the set $\mathbb{L}_k$ with l_k nodes.

For illustrative purposes, an example of such a system is depicted in Figure 2.1. In this example, there are three disconnected areas, which makes $p = 3$ and, consequently, $\mathbb{S} = \{\mathbb{S}_1, \mathbb{S}_2, \mathbb{S}_3\}$. In addition, four global links interconnect the three subsystems, making $l = 4$ and $\mathbb{L} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. As for the local quantities, subsystem $\mathbb{S}_1$ has $b_1 = 3$ and $l_1 = 3$; subsystem $\mathbb{S}_2$ has $b_2 = 2$ and $l_2 = 2$; and, subsystem $\mathbb{S}_3$ has $b_3 = 2$ and $l_3 = 3$.

Consider, now, that each subsystem $\mathbb{S}_k \in \mathbb{S}$ is modeled by the nodal equations given in (2.1), where $\mathbf{Y}_k$ is the admittance matrix of the isolated subsystem $\mathbb{S}_k$, $\mathbf{v}_k$ is a vector with nodal voltages, $\mathbf{j}_k$ a vector with internal injections and $\mathbf{i}_k$ a vector with currents injected by the local links $\mathbb{L}_k$. If the subsystem $\mathbb{S}_k$ has n_k *local buses*, $\mathbf{Y}_k$ is a square and, usually, sparse matrix with dimension n_k, whereas $\mathbf{v}_k$, $\mathbf{i}_k$ and $\mathbf{j}_k$ are vectors of order n_k. Additionally, the set of nodes in the subsystem $\mathbb{S}_k$ is defined as $\mathbb{N}_k$.

$$\mathbf{Y}_k\,\mathbf{v}_k = \mathbf{i}_k + \mathbf{j}_k \tag{2.2}$$

Usually, during power systems computations, e.g., dynamic simulations, the subsystems

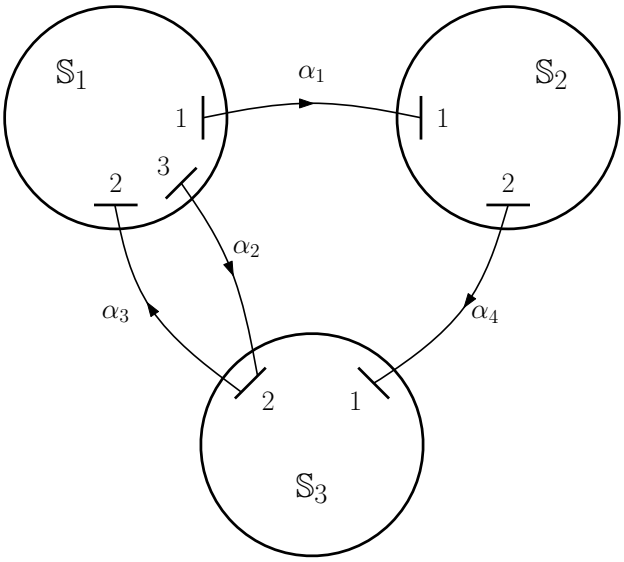


Figure 2.1. Generic electric network partitioned into three subsystems interconnected by four tie lines.

matrices $\mathbf{Y}_k$ and their associated local injections $\mathbf{j}_k$ are known, while the local nodal voltages $\mathbf{v}_k$ and local link injections $\mathbf{i}_k$ are unknown. Notice, however, that if the links contributions $\mathbf{i}_k$ were known, $\mathbf{v}_k$ would be straightforward to computed. Therefore, in the MATE context, the link currents $\mathbf{i}_k$, which depend on how the subsystems are connected to one another, need to be computed first and then submitted to their correspondent subsystems so local voltages $\mathbf{v}_k$ can be obtained.

2.2 MATE Original Formulation

Following the same reasoning presented in (Martí et al., 2002), the three-area electric system, depicted in Figure 2.1, can be mathematically represented by the hybrid modified nodal equations, given in (2.3). In other words, the latter model comprises nodal equations for the subsystems $\mathbb{S}_1$, $\mathbb{S}_2$ and $\mathbb{S}_3$, and branch equations for the links α_1, α_2, α_3 and α_4.

$$\left[\begin{array}{ccc|c} \mathbf{Y}_1 & & & \mathbf{P}_1 \\ & \mathbf{Y}_2 & & \mathbf{P}_2 \\ & & \mathbf{Y}_3 & \mathbf{P}_3 \\ \hline (\mathbf{P}_1)^T & (\mathbf{P}_2)^T & (\mathbf{P}_3)^T & -\mathbf{Z}_0^l \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \hline \mathbf{i}^l \end{array}\right] = \left[\begin{array}{c} \mathbf{j}_1 \\ \mathbf{j}_2 \\ \mathbf{j}_3 \\ \hline \mathbf{0} \end{array}\right] \tag{2.3}$$

The matrices $\mathbf{P}_k$, defined in (2.4), capture which nodes in subsystem $\mathbb{S}_k$ are connected to links, and whether each link is injecting/drawing current into/from that specific node.

$$\mathbf{P}_k(i,j) = \begin{cases} 1 & \text{if link } j \text{ injects current into bus } i, \\ -1 & \text{if link } j \text{ draws current from bus } i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

where i refers to buses in the subsystem $\mathbb{S}_k$ and j to global links.

In order to solve (2.3), one can apply Gaussian elimination to the row correspondent to the link currents $\mathbf{i}^l$, which yields the new set of equations given in (2.5).

$$\left[\begin{array}{ccc|c} \mathbf{Y}_1 & & & \mathbf{P}_1 \\ & \mathbf{Y}_2 & & \mathbf{P}_2 \\ & & \mathbf{Y}_3 & \mathbf{P}_3 \\ \hline & & & \mathbf{Z}^l \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \hline \mathbf{i}^l \end{array}\right] = \left[\begin{array}{c} \mathbf{j}_1 \\ \mathbf{j}_2 \\ \mathbf{j}_3 \\ \hline \mathbf{e}^l \end{array}\right] \tag{2.5}$$

Here, $\mathbf{Z}^l$ and $\mathbf{e}^l$ are defined below, with $k = 1, 2, 3$.

$$\mathbf{Z}^l = \mathbf{Z}_0^l + \sum_{\forall k} \mathbf{Z}_k^l \qquad\qquad \mathbf{e}^l = \sum_{\forall k} \mathbf{e}_k^l \tag{2.6}$$

$$\mathbf{Z}_k^l = (\mathbf{P}_k)^T (\mathbf{Y}_k)^{-1} \mathbf{P}_k \qquad\qquad \mathbf{e}_k^l = (\mathbf{P}_k)^T (\mathbf{Y}_k)^{-1} \mathbf{j}_k \tag{2.7}$$

In the transformed system of equations (2.5), each pair, $\mathbf{Z}_k^l$ and $\mathbf{e}_k^l$, represents the Thévenin equivalent of the subsystem $\mathbb{S}_k$ with respect to the links α_l with $l = 1, \ldots, 4$, whereas $\mathbf{Z}^l$ and $\mathbf{e}^l$ represent the reduced version of the original system from the links' perspective.

Ultimately, the original system can be solved by solving the link currents system (at the bottom of (2.5)) and then appropriately injecting the currents $\mathbf{i}^l$ into the subsystems $\mathbb{S}_k$ with $k = 1, 2, 3$.

As pointed by Martí et al. (2002), the main advantage of the MATE algorithm lies in the fact that subsystems can be computed independently from each other, apart from the links system of equations. This fact, in turn, leads to other possibilities, such as: subsystems can be solved concurrently at different rates, with different integration methods, or even in different domains (like time and quasi-stationary phasor domains). The possibility of parallel computations at the subsystems level has been conceived in (Martí et al., 2002; Armstrong et al., 2006), but only implemented on a distributed computer architecture for

solving nodal equations of large electric power systems in (Tomim et al., 2008).

Recent investigations of the MATE algorithm, however, indicated that the set of equations (2.5), (2.6) and (2.7) hide a few interesting features that help reduce amount of operations performed in the subsystems and the data exchange between link solver and subsystems. Therefore, in the subsequent sections, the MATE algorithm will come under close scrutiny in order to make such features evident.

2.3 Network-based MATE Formulation

Even though the set of equations given in (2.5) are sufficient to describe the algorithm under study, they hide a few interesting features that help reduce the number of operations performed in the subsystems and the amount of data exchange between link solver and subsystems. Therefore, in the next sections, the MATE algorithm will be restated from an electric network viewpoint in order to make these features evident.

2.3.1 MATE Algorithm Summary

For solving the problem stated in Section 2.1, one could start by extracting the links from the original system $\mathbb{S}$ and finding a multi-node Thévenin equivalent with respect to the border nodes, as suggested in (Dommel, 1996). Since the subsystems are uncoupled *a priori*, the Thévenin equivalents computation could also be done for each subsystem $\mathbb{S}_k$ separately. The multi-node Thévenin equivalent of each subsystem, as seen from its border nodes, can be constructed in two basic steps:

(a) Find the voltages at the border nodes due to the internal current and voltage sources;

(b) Find the self and mutual impedances seen from the border nodes (short-circuiting and opening internal voltage and current sources, respectively).

As a result of the previous steps, reduced-order networks, completely isolated from each other, will be generated for each subsystem $\mathbb{S}_k$. These equivalent networks, along with the interconnection system, will form a reduced-order version of the original system, as seen from the links. The currents flowing in each link α_i, with $i = 1, \ldots, l$, can then be computed employing the newly constructed equivalent system.

Once the currents flowing in the links are known, each vector $\mathbf{i}_k$ can be formed according to the links connections to the subsystems. Lastly, combining $\mathbf{i}_k$ vectors with the internal voltage and current sources represented by $\mathbf{j}_k$ enables the computation of each subsystems nodal voltages $\mathbf{v}_k$ using (2.1).

In the next sections, each MATE algorithm building block will be explained in more detail.

2.3.2 Multi-Node Thévenin Equivalents

The first step in the MATE algorithm is constructing multi-node Thévenin equivalents for all subsystems $\mathbb{S}_k \in \mathbb{S}$ with respect to their border nodes set $\mathbb{B}_k$. Such equivalent circuits fully describe the voltage-current characteristic of each subsystem at its border nodes, and will provide the fundamental structures for solving the link currents at a later stage. In addition, as an aid to the multi-node Thévenin equivalents construction, a subsystem-to-border mapping will also be introduced.

Subsystem-to-Border Mapping

Given a specific subsystem $\mathbb{S}_k \in \mathbb{S}$, it will be necessary to map its b_k border nodes quantities onto its n_k local buses quantities, i.e., map quantities defined in $\mathbb{B}_k$ onto $\mathbb{N}_k$. One such situation occurs when one needs to express current injections defined in the border nodes set $\mathbb{B}_k$ in terms of the local nodes set $\mathbb{N}_k$. The inverse mapping, i.e., mapping of $\mathbb{N}_k$ onto $\mathbb{B}_k$, is equally important, since it allows one to gather border nodes voltages from local nodal voltages.

Formally, the mapping of $\mathbb{N}_k$ onto $\mathbb{B}_k$ can be seen as a $b_k \times n_k$ matrix $\mathbf{Q}_k$ defined in (2.8). The matrix $\mathbf{Q}_k$ is filled with zeros, except for b_k ones, which indicate that a specific local node j (related to the columns of $\mathbf{Q}_k$) corresponds to the border node i (related to the rows of $\mathbf{Q}_k$).

$$\mathbf{Q}_k(i,j) = \begin{cases} 0 & \text{if } \mathbb{B}_k(i) \neq \mathbb{N}_k(j), \\ 1 & \text{if } \mathbb{B}_k(i) = \mathbb{N}_k(j). \end{cases} \tag{2.8}$$

where $i = 1, \ldots, b_k$ and $j = 1, \ldots, n_k$.

As for the inverse direction of the mapping, it is achieved by $(\mathbf{Q}_k)^T$. This inverse mapping simply scatters the vectors defined in the border node set $\mathbb{B}_k$ so they are expressed in terms of the local nodes set $\mathbb{N}_k$. Although the vectors produced by $(\mathbf{Q}_k)^T$ have n_k elements, only b_k of them are non-zero, which correspond to the border nodes.

Thus, if the vector $\mathbf{v}_k$ contains voltages associated with the set of local nodes $\mathbb{N}_k$, gathering border nodes voltages $\mathbf{v}_k^b$ from $\mathbf{v}_k$ can be summarized as the matrix-vector multiplication expressed by (2.9).

$$\mathbf{v}_k^b = \mathbf{Q}_k \, \mathbf{v}_k \tag{2.9}$$

On the other hand, consider now a vector $\mathbf{i}_k^b$ that contains current injections associated

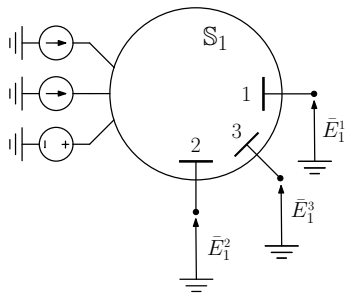


Figure 2.2. Thévenin equivalent voltages.

with the set of border nodes $\mathbb{B}_k$. If one wants to scatter the injections $\mathbf{i}_k^b$ into another vector, say $\mathbf{i}_k$, defined in $\mathbb{N}_k$, one only needs the matrix-vector multiplication expressed by (2.10).

$$\mathbf{i}_k = (\mathbf{Q}_k)^T \, \mathbf{i}_k^b \tag{2.10}$$

For example, for the subsystem $\mathbb{S}_1$ of the system depicted in Figure 2.1, the matrix $\mathbf{Q}_1$ is shown in (2.11), assuming that the set of border nodes $\mathbb{B}_1$ contains $b_1 = 3$ nodes.

$$\mathbf{Q}_1 = \begin{array}{c} \textbf{system nodes} \\ \begin{matrix} \textcircled{1} & \cdots & \textcircled{3} & \cdots & \textcircled{2} & \cdots \end{matrix} \\ \begin{bmatrix} 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & 1 & \cdots \\ \cdots & \cdots & 1 & \cdots & \cdots & \cdots \end{bmatrix} \end{array} \begin{matrix} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} \end{matrix} \textbf{border} \tag{2.11}$$

Multi-Node Thévenin Voltage

Multi-node Thévenin voltages are nodal voltages detected at the border nodes of a specific subsystem, due to internal voltage and current sources only. In such a situation, all links connected to the same subsystem must be open and all subsystem internal sources active during the computation of border nodal voltages. The Figure 2.2 illustrates the concept of multi-node Thévenin voltages when applied to subsystem $\mathbb{S}_1$, depicted in Figure 2.1. Mathematically, the multi-node Thévenin voltages computation can be summarized by the two step procedure shown in (2.12).

$$\mathbf{Y}_k \, \mathbf{e}_k = \mathbf{j}_k \tag{2.12a}$$

$$\mathbf{e}_k^b = \mathbf{Q}_k \, \mathbf{e}_k \tag{2.12b}$$

First, by solving the linear system (2.12a), the new voltage vector $\mathbf{e}_k$ is obtained, which

contains Thévenin voltages at all nodes of the subsystem $\mathbb{S}_k$ described by $\mathbf{Y}_k$. However, since only Thévenin voltages at the border nodes are needed, one needs to gather the corresponding voltages in $\mathbf{e}_k$ and store them in $\mathbf{e}_k^b$ by means of the subsystem-to-border mapping matrix $\mathbf{Q}_k$, according to (2.12b).

Multi-Node Thévenin Impedance

Multi-node Thévenin impedances are a set of self and mutual impedances seen from the border nodes of a specific subsystem $\mathbb{S}_k$, when all its internal voltage and current sources are turned off, i.e., the voltage sources become short circuits and the current sources open circuits. For compactness, it can also be represented as a $b_k \times b_k$ matrix $\mathbf{Z}_k^b$ which relates the injected currents at the border nodes $\mathbf{i}_k^b$ to the border nodes voltages $\mathbf{v}_k^b$.

$$\mathbf{v}_k^b = \mathbf{Z}_k^b \mathbf{i}_k^b \tag{2.13}$$

In the general case, the matrix $\mathbf{Z}_k^b$ can be obtained column by column, injecting unitary currents into its corresponding border node and measuring the voltages at all border nodes. Thus, bearing in mind the fact that each subsystem $\mathbb{S}_k$ is represented by its nodal equations, voltages at the border nodes can be gathered from repeated solutions of (2.1) for individual current injections at each border node. In turn, each unitary current injection can be represented by a single column vector $\mathbf{u}_k^{bi}$ of size b_k, which has one 1 at the position i, associated with the border node $\mathbb{B}_k(i)$, while all other components are zero. Now, taking into consideration that $\mathbf{Y}_k$ is related to the local nodes in $\mathbb{N}_k$, one first needs to map $\mathbf{u}_k^{bi}$ onto $\mathbb{N}_k$, using the border-to-subsystem mapping $(\mathbf{Q}_k)^T$. So, collecting all the vectors $\mathbf{u}_k^{bi}$ associated with $i = 1, \ldots, b_k$, in this order, and applying the transformation $(\mathbf{Q}_k)^T$ to it, leads to (2.14). This shows that the unitary current injections required for computing $\mathbf{Z}_k^b$ are readily provided by $(\mathbf{Q}_k)^T$.

$$(\mathbf{Q}_k)^T \begin{bmatrix} \mathbf{u}_k^{b1} & \mathbf{u}_k^{b2} & \cdots & \mathbf{u}_k^{bb_k} \end{bmatrix} = (\mathbf{Q}_k)^T \tag{2.14}$$

$$(\mathbf{Q}_k)^T \begin{bmatrix} \mathbf{u}_k^{b1} & \mathbf{u}_k^{b2} & \cdots & \mathbf{u}_k^{bb_k} \end{bmatrix} = (\mathbf{Q}_k)^T \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = (\mathbf{Q}_k)^T \tag{2.15}$$

Hence, a general procedure for computing $\mathbf{Z}_k^b$, with respect to border nodes of $\mathbb{S}_k$ is summarized by (2.16). Since $(\mathbf{Q}_k)^T$ has b_k columns, the equation (2.16a) is equivalent to

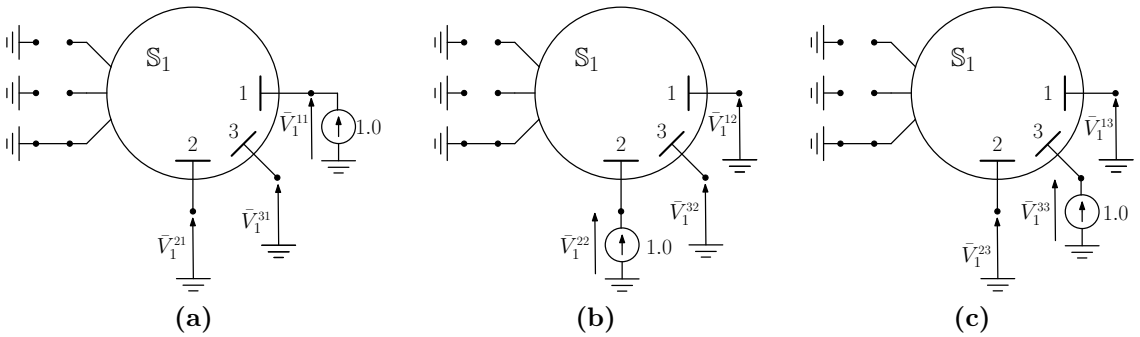


Figure 2.3. Multinode Thévenin equivalent impedances construction.

solving a sparse linear system of order n_k, defined by $\mathbf{Y}_k$, b_k times, and storing each solution in each column of $\mathbf{Z}_k$. Afterwards, one only needs to gather the impedances associated with the border nodes in $\mathbf{Z}_k$ and store them in $\mathbf{Z}_k^b$, as stated by (2.16b).

$$\mathbf{Y}_k \, \mathbf{Z}_k = (\mathbf{Q}_k)^T \tag{2.16a}$$

$$\mathbf{Z}_k^b = \mathbf{Q}_k \, \mathbf{Z}_k \tag{2.16b}$$

Figure 2.3 illustrates the concept of multi-node Thévenin voltages when applied to subsystem $\mathbb{S}_1$, depicted in Figure 2.1.

Multi-Node Thévenin Equivalents

After equations (2.12) and (2.16) were solved for each subsystem $\mathbb{S}_k \in \mathbb{S}$, a series of circuits, synthesized as shown in Figure 2.4, is constructed. Algebraically, the multi-node Thévenin equivalent can be represented by (2.17), where $\mathbf{i}_k^b$ represent the current injections at the border nodes, $\mathbf{v}_k^b$ lumps all the voltage drops across the subsystem $\mathbb{S}_k$, and $\mathbf{e}_k^b$ corresponds to the vector with the Thévenin voltages that lie behind the Thévenin impedance network $\mathbf{Z}_k^b$.

$$\mathbf{v}_k^b = \mathbf{Z}_k^b \mathbf{i}_k^b + \mathbf{e}_k^b \tag{2.17}$$

$$\mathbf{v}_k^b = \begin{bmatrix} \bar{V}_k^1 - \bar{V}_k^{1'} \\ \vdots \\ \bar{V}_k^i - \bar{V}_k^{i'} \\ \vdots \\ \bar{V}_k^{b_k} - \bar{V}_k^{b'_k} \end{bmatrix} \qquad \mathbf{i}_k^b = \begin{bmatrix} \bar{I}_k^1 \\ \vdots \\ \bar{I}_k^i \\ \vdots \\ \bar{I}_k^{b_k} \end{bmatrix} \qquad \mathbf{e}_k^b = \begin{bmatrix} \bar{E}_k^1 \\ \vdots \\ \bar{E}_k^i \\ \vdots \\ \bar{E}_k^{b_k} \end{bmatrix}$$

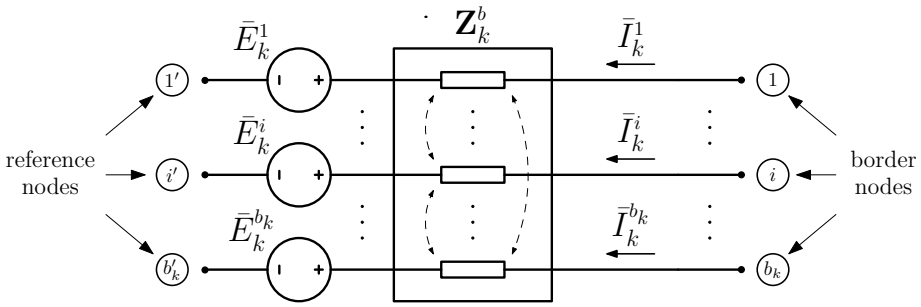


Figure 2.4. Multi-node Thévenin equivalent of a generic subsystem $\mathbb{S}_k$ with b_k border nodes.

2.3.3 Multi-Area Thévenin Equivalents

Once all subsystems $\mathbb{S}_k \in \mathbb{S}$ have been reduced to their multi-node Thévenin equivalents, one needs to interconnect them through the link system so the currents flowing between adjacent subsystems can be calculated. This final equivalent of the original system with respect to all links is then called the Multi-Area Thévenin Equivalent (MATE). Analogously to what has been done for the multi-node Thévenin equivalents construction, a link-to-border mapping will also be introduced.

Link-to-Border Mapping

The aim of the link-to-border mapping is to represent current injections at the border nodes $\mathbb{B}_k$ in terms of currents flowing in the links defined in the set of global links $\mathbb{L}$. For instance, for the subsystem $\mathbb{S}_1$ in Figure 2.1 which has $b_1 = l_1 = 3$, the current injections at each border node can be seen as an injection coming from a single link, respecting proper link current orientation. Thus, if $\bar{I}_k^i$ is a current injection at border node $\mathbb{B}_k(i)$, $\bar{I}_A^1 = -\bar{I}^{\alpha_1}$, $\bar{I}_A^2 = \bar{I}^{\alpha_3}$, and $\bar{I}_A^3 = -\bar{I}^{\alpha_2}$. In matrix form, the previous relationships yield (2.18), where $\mathbf{i}_1^b$ is a vector with the border nodes injections, $\mathbf{i}^l$ is a vector with the link currents according to the orientation adopted in Figure 2.1, and $\mathbf{R}_1$ is a $b_1 \times l$ matrix that represents the mapping between the two vectors.

$$\mathbf{i}_1^b = \mathbf{R}_1 \mathbf{i}^l \tag{2.18}$$

where,

$$\mathbf{i}_1^b = \begin{bmatrix} \bar{I}_1^1 \\ \bar{I}_1^2 \\ \bar{I}_1^3 \end{bmatrix} \qquad \mathbf{i}^l = \begin{bmatrix} \bar{I}^{\alpha_1} \\ \bar{I}^{\alpha_2} \\ \bar{I}^{\alpha_3} \\ \bar{I}^{\alpha_4} \end{bmatrix} \qquad \mathbf{R}_1 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

For the subsystem $\mathbb{S}_3$ in the same example, $b_3 = 2$, $l_3 = 3$, and the current injections at each border node become combinations of one or more links. In this case, $\bar{I}_3^1 = \bar{I}^{\alpha_4}$ and $\bar{I}_3^2 = \bar{I}^{\alpha_2} - \bar{I}^{\alpha_3}$, which leads to (2.19).

$$\mathbf{i}_3^b = \mathbf{R}_3 \mathbf{i}^l \tag{2.19}$$

where,

$$\mathbf{i}_3^b = \begin{bmatrix} \bar{I}_3^1 \\ \bar{I}_3^2 \end{bmatrix} \qquad \mathbf{R}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

So far, only the current injections were related by means of the link-to-border mapping. In the case of the voltages, the mapping direction has to be reversed. This can be explained by the fact that each local link connecting the same border node senses the same nodal voltage. However, the link orientation may change the polarity of the voltage. This is the case in subsystem $\mathbb{S}_3$ in Figure 2.1, where links α_2 and α_3 are joined together at the border node $\mathbb{B}_1(2)$. Thus, if $\bar{V}_k^i$ is a nodal voltage at border node $\mathbb{B}_k(i)$, and $\bar{V}_k^{\alpha_j}$ the voltage at link α_j whose termination is in the subsystem $\mathbb{S}_k$, equation (2.20) applies.

$$\mathbf{v}_3^l = (\mathbf{R}_3)^T \mathbf{v}_3^b \tag{2.20}$$

where,

$$\mathbf{v}_3^b = \begin{bmatrix} \bar{V}_3^1 \\ \bar{V}_3^2 \end{bmatrix} \qquad \mathbf{v}_3^l = \begin{bmatrix} \bar{V}_3^{\alpha_1} \\ \bar{V}_3^{\alpha_2} \\ \bar{V}_3^{\alpha_3} \\ \bar{V}_3^{\alpha_4} \end{bmatrix} \qquad (\mathbf{R}_3)^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & -1 \\ 1 & 0 \end{bmatrix}$$

In a general form, the mapping $\mathbf{R}_k$, from $\mathbb{L}$ onto $\mathbb{B}_k$, can be defined by (2.21). Conversely, border-to-link mapping is achieved by employing $(\mathbf{R}_k)^T$.

$$\mathbf{R}_k(i,j) = \begin{cases} 1 & \text{if } \mathbb{L}(j) \text{ injects current into } \mathbb{B}_k(i), \\ -1 & \text{if } \mathbb{L}(j) \text{ draws current from } \mathbb{B}_k(i), \\ 0 & \text{otherwise.} \end{cases} \tag{2.21}$$

where $i = 1, \ldots, b_k$ and $j = 1, \ldots, l$.

Note that $\mathbf{R}_k$ is sparse and each row, associated with the border nodes of subsystem $\mathbb{S}_k$, has as many non-zero entries as the number of links connected to it. Therefore, the number

of non-zeros in $\mathbf{R}_k$ always equals the number of local links in $\mathbb{L}_k$, i.e., l_k.

Link Thévenin Equivalents

At this point, all subsystems $\mathbb{S}_k \in \mathbb{S}$ are reduced to their set of border nodes $\mathbb{B}_k$ by means of multi-node Thévenin equivalents and can be finally reconnected through the link system. Since the equation (2.17) is expressed in terms of the border nodes in $\mathbb{B}_k$, each multi-node Thévenin equivalent has to be referenced to the set of global links $\mathbb{L}$, so all equivalents can be connected to one another. The latter equivalents are then called *link Thévenin equivalents*, and will be discussed next.

Recalling that each vector $\mathbf{v}_k^b$ and $\mathbf{e}_k^b$, with border nodes voltages and Thévenin voltages, respectively, can be mapped onto $\mathbb{L}$ by means of $(\mathbf{R}_k)^T$, and the link currents vector $\mathbf{i}^l$ mapped onto each $\mathbb{B}_k$ by means of the mapping $\mathbf{R}_k$, one can pre-multiply (2.17) by $(\mathbf{R}_k)^T$ and make $\mathbf{i}_k^b = \mathbf{R}_k \mathbf{i}^l$, as follows:

$$(\mathbf{R}_k)^T \mathbf{v}_k^b = \left[(\mathbf{R}_k)^T \mathbf{Z}_k^b \mathbf{R}_k\right] \mathbf{i}^l + (\mathbf{R}_k)^T \mathbf{e}_k^b$$

which yields:

$$\mathbf{v}_k^l = \mathbf{Z}_k^l \mathbf{i}^l + \mathbf{e}_k^l \tag{2.22a}$$

$$\mathbf{Z}_k^l = (\mathbf{R}_k)^T \mathbf{Z}_k^b \mathbf{R}_k \tag{2.22b}$$

$$\mathbf{e}_k^l = (\mathbf{R}_k)^T \mathbf{e}_k^b \tag{2.22c}$$

In the above equations, $\mathbf{e}_k^l$ represents the vector with the Thévenin voltages associated with subsystem $\mathbb{S}_k$ that partially drive the global link currents $\mathbf{i}^l$. The vector $\mathbf{v}_k^l$ contains the total voltage drops across the subsystem $\mathbb{S}_k$ due to $\mathbf{i}^l$ and $\mathbf{e}_k^l$. Lastly, the $l \times l$ matrix $\mathbf{Z}_k^l$, given in (2.22b), is the set of impedances $\mathbf{Z}_k^b$ seen from the global links.

For example, applying link-to-border mapping $\mathbf{R}_2$ to the multi-node Thévenin equivalent $\mathbf{Z}_2^b$ with respect to the border nodes of subsystem $\mathbb{S}_2$ in Figure 2.1, as given in (2.23), one can obtain $\mathbf{e}_2^l$, $\mathbf{v}_2^l$ and $\mathbf{Z}_2^l$, given in (2.24), which form the link Thévenin equivalent of subsystem $\mathbb{S}_2$.

$$\mathbf{Z}_2^b = \begin{bmatrix} \bar{Z}_2^{11} & \bar{Z}_2^{12} \\ \bar{Z}_2^{21} & \bar{Z}_2^{22} \end{bmatrix} \qquad \mathbf{R}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{2.23}$$

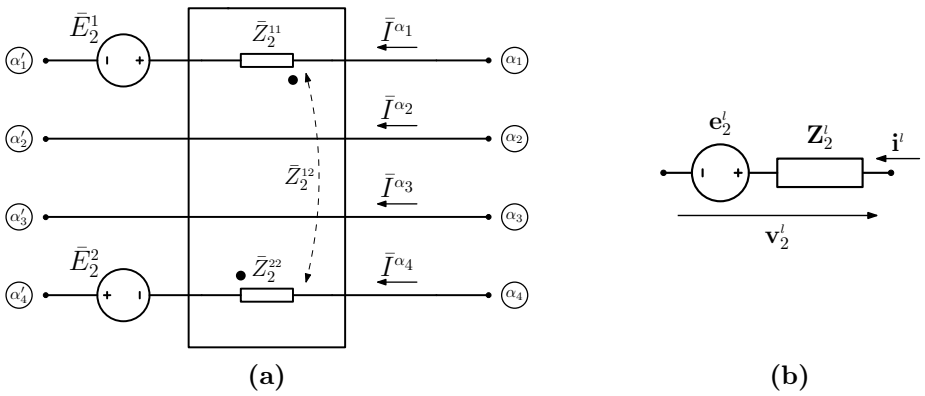


Figure 2.5. Link Thévenin equivalent of subsystem $\mathbb{S}_2$ in Figure 2.1 in detailed and compact forms, respectively.

$$\mathbf{e}_2^l = (\mathbf{R}_2)^T \mathbf{e}_2^b = \begin{bmatrix} \bar{E}_2^{\alpha_1} \\ \bar{E}_2^{\alpha_2} \\ \bar{E}_2^{\alpha_3} \\ \bar{E}_2^{\alpha_4} \end{bmatrix} = \begin{bmatrix} \bar{E}_2^1 \\ 0 \\ 0 \\ -\bar{E}_2^2 \end{bmatrix} \tag{2.24a}$$

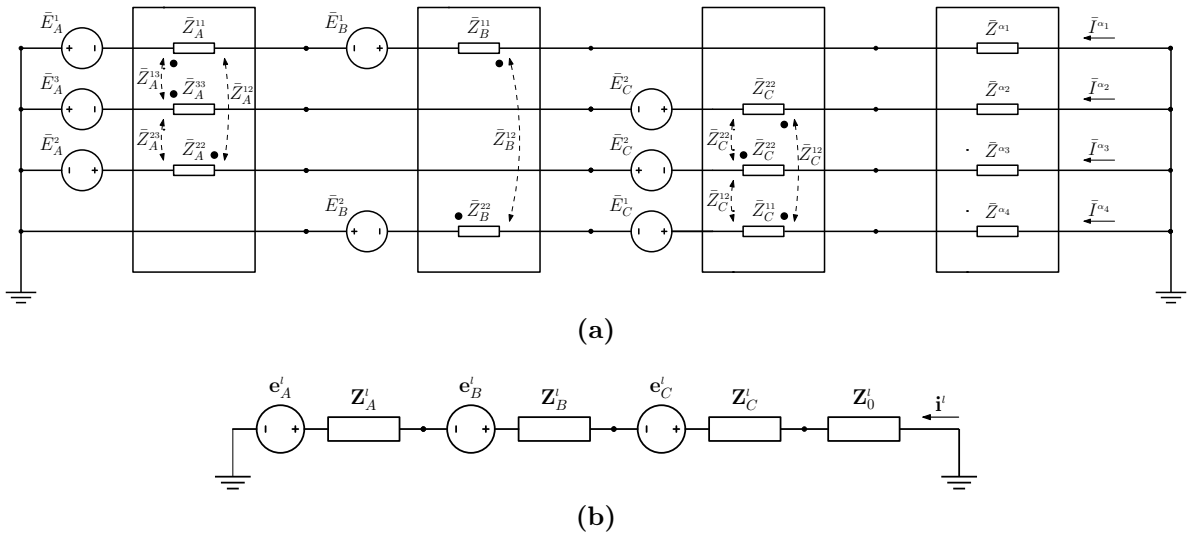
$$\mathbf{v}_2^l = (\mathbf{R}_2)^T \mathbf{v}_2^b = \begin{bmatrix} \bar{V}_2^{\alpha_1} \\ \bar{V}_2^{\alpha_2} \\ \bar{V}_2^{\alpha_3} \\ \bar{V}_2^{\alpha_4} \end{bmatrix} = \begin{bmatrix} \bar{V}_2^1 \\ 0 \\ 0 \\ -\bar{V}_2^2 \end{bmatrix} \tag{2.24b}$$

$$\mathbf{Z}_2^l = (\mathbf{R}_2)^T \mathbf{Z}_2^b \mathbf{R}_2 = \begin{bmatrix} \bar{Z}_2^{11} & 0 & 0 & -\bar{Z}_2^{12} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\bar{Z}_B^{21} & 0 & 0 & \bar{Z}_B^{22} \end{bmatrix} \tag{2.24c}$$

As matter of fact, equation (2.22) can be seen as a polyphase voltage source $\mathbf{e}_k^l$ in series with the polyphase impedance $\mathbf{Z}_k^l$. Thus, the link Thévenin equivalent of the subsystem $\mathbb{S}_2$, described by (2.24), can be represented by the equivalent circuits shown in Figure 2.5.

Multi-Area Thévenin Equivalents

Once the link Thévenin equivalents are found for all subsystems $\mathbb{S}_k \in \mathbb{S}$, for each subsystem there will exist a polyphase representation similar to the one shown in Figure 2.5. Since each phase of each link Thévenin equivalent is associated with one single global link, they can all



(a)

(b)

Figure 2.6. Multi-area Thévenin equivalent of the system in Figure 2.1 in its detailed and compact versions, respectively.

be connected in series as shown in Figure 2.6.

In addition to the subsystems, one can also add the impedances of the interconnection system, represented by the matrix $\mathbf{Z}_0^l$. In the case of the system example depicted in Figure 2.6, $\mathbf{Z}_0^l$ is a diagonal matrix, because all links are considered uncoupled. In a more general case, however, couplings among links could be straightforwardly added as off-diagonal elements of $\mathbf{Z}_0^l$.

Finally, based on the circuit Figure 2.6b, one can combine all polyphase impedances and voltages sources accordingly, which results in the linear system given in (2.25), which can then be solved for the link currents $\mathbf{i}^l$.

$$\mathbf{Z}^l = \mathbf{Z}_0^l + \sum_{\mathbb{S}_k \in \mathbb{S}} \mathbf{Z}_k^l \tag{2.25a}$$

$$\mathbf{e}^l = \sum_{\mathbb{S}_k \in \mathbb{S}} \mathbf{e}_k^l \tag{2.25b}$$

$$\mathbf{Z}^l \mathbf{i}^l = \mathbf{e}^l \tag{2.25c}$$

2.3.4 Subsystems Update

Since all link currents $\mathbf{i}^l$ are known at this stage, one only needs to build the proper $\mathbf{i}_k$ from the link currents $\mathbf{i}^l$ and, together with the internal currents $\mathbf{j}_k$, solve the linear system (2.1) for the nodal voltages $\mathbf{v}_k$ for each subsystem $\mathbb{S}_k$.

For building $\mathbf{i}_k$, a two step conversion is proposed, first from links to border nodes and

then from border nodes to subsystem nodes. Mathematically, this task is achieved by means of the transformations $\mathbf{R}_k$ and $(\mathbf{Q}_k)^T$, defined in (2.21) and (2.8), respectively, and expressed in (2.26).

$$\mathbf{i}_k = (\mathbf{Q}_k)^T \mathbf{R}_k \mathbf{i}^l \tag{2.26}$$

By the end of this stage, each subsystem $\mathbb{S}_k \in \mathbb{S}$ will have its set of nodal voltages $\mathbf{v}_k$ known, which completely solves the original untorn system.

2.4 MATE: Original versus Network-based

The difference between the original MATE formulation, introduced by Martí et al. (2002), and the network-based formulation, presently discussed, lies on the definitions of the subsystem-to-border mapping $\mathbf{Q}_k$, given in (2.8), and the link-to-border mapping $\mathbf{R}_k$, given in (2.21), and how they relate to $\mathbf{P}_k$, given in (2.3).

From (2.3), notice that $\mathbf{P}_k$ converts the global link currents $\mathbf{i}^l$ into currents injected into the subsystem $\mathbb{S}_k$. Therefore, each matrix $\mathbf{P}_k$ can be seen as a link-to-subsystem mapping from $\mathbb{L}$ to $\mathbb{N}_k$, as defined in (2.4) and emphasized in (2.27).

$$\mathbf{i}_k = \mathbf{P}_k \mathbf{i}^l \tag{2.27}$$

Comparing then (2.27) to (2.26), one can verify that the following equality applies.

$$\mathbf{P}_k = (\mathbf{Q}_k)^T \mathbf{R}_k \tag{2.28}$$

Similarly to the other mappings, the reverse mapping of $\mathbf{P}_k$ is also given by its transposed version, i.e., $(\mathbf{P}_k)^T$, and can be used to convert the nodal voltages defined in $\mathbb{N}_k$ to voltages at links terminals in $\mathbb{L}$.

The relationships among all defined mappings are summarized in Figure 2.7, which includes the flow directions of node voltages and current injections among subsystem nodes set $\mathbb{N}_k$, border nodes set $\mathbb{B}_k$ and global links set $\mathbb{L}$.

The main benefits in using $\mathbf{R}_k$ and $\mathbf{Q}_k$ mappings instead of $\mathbf{P}_k$ are the following:

Less computational effort to obtain the multi-area Thévenin equivalent $\mathbf{Z}^l$: The multi-area Thévenin equivalent $\mathbf{Z}^l$ can be interchangeably defined in (2.25a) and (2.6). However, if each link Thévenin equivalent $\mathbf{Z}^l_k$, which will later form the multi-area Thévenin equivalent $\mathbf{Z}^l$, is computed according (2.7), the nodal equations associated with the subsystem $\mathbb{S}_k$ need to be solved l times, where l is the number of global links. Even when the

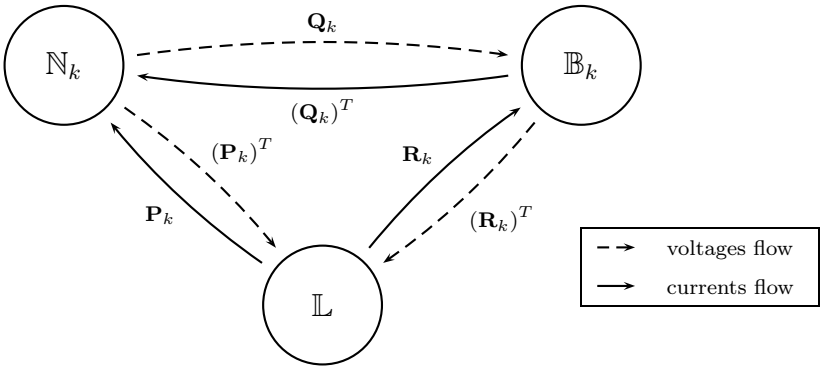


Figure 2.7. Relationships among the various mappings $\mathbf{R}_k$, $\mathbf{Q}_k$ and $\mathbf{P}_k$.

empty columns of $\mathbf{P}_k$ are skipped during the computation of $\mathbf{Z}_k^l$, the subsystem $\mathbb{S}_k$ needs to be solved at least l_k times, where l_k is the number of local links. On the other hand, if each link Thévenin equivalent $\mathbf{Z}_k^l$ is computed according to (2.22b), the computational burden will be concentrated in forming the multi-node Thévenin equivalents $\mathbf{Z}_k^b$, which requires only b_k solutions for each subsystem $\mathbb{S}_k$. Taking into consideration that, for large systems, the number of border nodes $b_k \leq l_k \ll l$, a reasonable gain in performance should be expected.

Less data exchange between subsystems processes and link solver: The link-to-border mapping $\mathbf{R}_k$ works as an indirect reference to the multi-node Thévenin equivalent $\mathbf{Z}_k^b$, which has only ${b_k}^2$ elements, to form the link Thévenin equivalent $\mathbf{Z}_k^l$, which in turn has ${l_k}^2$ elements. Without the information inherited by the link-to-border mapping $\mathbf{R}_k$, $\mathbf{Z}_k^l$ would have to be formed by the subsystems and sent to the link solver. Moreover, without $\mathbf{R}_k$, the link solver would have no other option, but sending all link currents to all subsystems $\mathbb{S}_k \in \mathbb{S}$. With $\mathbf{R}_k$, on the other hand, the link solver only need to send the net currents injected at the border nodes of the subsystems. Therefore, the link-to-border mapping $\mathbf{R}_k$ plays an important role in minimizing data exchange between subsystems.

Global link information is confined in the link solver process: If link Thévenin equivalents $\mathbf{Z}_k^l$ are formed by its subsystem processes, information regarding the global links is required by all subsystems and, therefore, needs to be replicated in all processes participating in the solution. The multi-node Thévenin equivalents $\mathbf{Z}_k^b$, however, only require information regarding the local links, reducing again memory usage and amount of global information that needs to be shared among the processes.

2.5 Conclusion

In this chapter, a step-by-step formulation of the MATE algorithm from a network standpoint has been developed. The formulation revealed properties of the subsystems interface nodes (or, border nodes) and the links that can be used to minimize both subsystems computations and communication overhead.

The distinct mappings among subsystem nodes, border nodes and links, introduced in this formulation, also make subsystem computations and communications rely on local information only. This characteristic also plays an important role in the aforementioned communication and computation reduction.

In the next chapter, a thorough assessment of the computation and communication effort required by the foregoing methodology, in terms of a performance model of the network-based MATE algorithm, will be presented.

Chapter 3

Network-based MATE Algorithm Implementation

Although diakoptics theory is very well established in literature, not many implementations of such techniques are available in power systems field. This fact is likely due to the great success of sparsity techniques in solving power systems problems in sequential computers, and high costs involved with early parallel machines.

However, with the advent of cheaper commodity computer clusters and multi-core processors, diakoptics-based as well as other parallel algorithms have been drawing more attention from power systems field. Therefore, decision makers need appropriate tools to compare existing traditional software to newly developed parallel alternatives. This requires an understanding of the related computational costs, which can be developed and formalized in mathematical performance models. These models can be used to compare the efficiency of different algorithms, to evaluate scalability, and to identify possible bottlenecks and other inefficiencies, all before investing substantial effort in an implementation (Foster, 1995).

Even though many parallel algorithms have been proposed in computer science and power systems fields for solving large sparse linear systems (see Chapter 1 for references), their performance greatly varies according to the available computing architecture and, the type and size of the problem under analysis. Depending on the target architecture, the maximum communication overhead due to the parallelization, while assuring reasonably efficient parallel computations, may drastically decrease, which is the case of distributed systems. As a consequence, fine grain parallel algorithms may be not be recommendable or even become unfeasible.

An implementation of the MATE algorithm suitable for distributed computing architectures will be set forth, according to the theoretical development presented in Chapter 2. One of the reasons supporting this choice comes from the fact that, in the MATE algorithm, the amount of communications remains limited, as long as the system under analysis is partitioned in so many subsystems, so the interconnections among them also remains limited.

In the development, the proposed implementation will be also modeled in terms of performance metrics, such as computation and communication times, efficiency and speedup. For

this task, the implementation will be assessed from the computational and communication standpoint, aiming at timings required to compute the subsystems and link equations, and to exchange data between subsystems and link solver processes. Afterwards, the aforementioned performance metrics for the MATE parallel algorithm, relative to sequential sparsity techniques, will be evaluated and analyzed.

3.1 MATE Algorithm Flow Chart

In Figure 3.1, the proposed MATE algorithm flow chart is depicted. Since all subsystems are similar in behavior, only one generic subsystem $\mathbb{S}_k$ is represented. Moreover, each communication point between the subsystem $\mathbb{S}_k$ and the link solver should actually be seen as a collective communication point, where all subsystems $\mathbb{S}_k \in \mathbb{S}$ communicate with the link solver and vice-versa.

At the beginning, all processes need to acquire the data they rely upon. At this stage, all subsystems concurrently load local admittance matrices $\mathbf{Y}_k$, local current injections $\mathbf{j}_k$, and information regarding the links connected to this specific subsystem, which is summarized by $\mathbb{L}_k$. As for the link solver, information about all links interconnecting all subsystems, denoted by simply $\mathbb{L}$, need to be loaded. Subsequently, subsystems identify the border nodes sets $\mathbb{B}_k$ and form the subsystem-to-border mappings $\mathbf{Q}_k$ and the link-to-border mappings $\mathbf{R}_k$, according to the definitions shown in (2.8) and (2.21). Afterwards, the link-to-border mappings $\mathbf{R}_k$ are sent to the link solver, so it becomes aware of which links are associated with each subsystem $\mathbb{S}_k$, and their respective border nodes. At this point all subsystems start computing their multi-node Thévenin equivalents, formed by $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$, according to (2.12), (2.16) and (2.17), and send them to the link solver. After receiving the previous equivalents, the link solver assembles the multi-area Thévenin equivalent, formed by $\mathbf{Z}^l$ and $\mathbf{e}^l$, and computes the link currents $\mathbf{i}^l$, following the procedure described by (2.22) and (2.25). With the link currents $\mathbf{i}^l$ available, the link solver scatters them appropriately, i.e., respecting the link-to-border mappings $\mathbf{R}_k$, among the subsystems, which can, finally, update the local injections $\mathbf{j}_k$ with the links coming from the link solver and compute the local node voltages $\mathbf{v}_k$.

In the next sections, a more elaborate discussion of the forgoing procedure will presented, along with performance aspects of each task involved in the network-based MATE implementation.

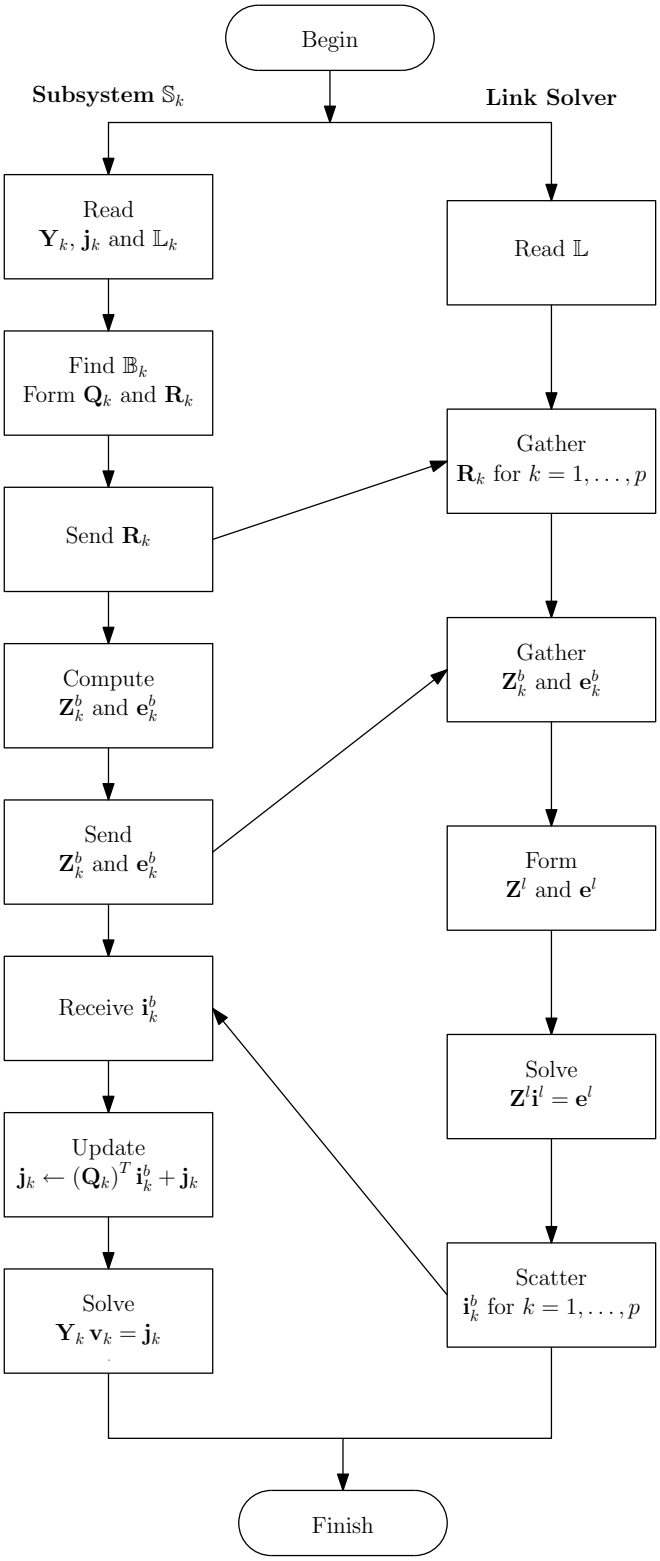


Figure 3.1. MATE algorithm flow chart

3.2 MATE Performance Model

In this section, the MATE method for solving large linear electric networks in a distributed computer architecture will be modeled in terms of performance metrics, such as computation and communication times, efficiency and speedup. For this task, the algorithm presented earlier will be assessed from the computational and communication standpoint, aiming at timings required to compute the subsystems and link equations, and to exchange data between subsystems and link solver processes. Afterwards, the aforementioned performance metrics for the MATE parallel algorithm, relative to sequential sparsity techniques, will be evaluated and analyzed.

3.2.1 Performance Model Preliminaries

In order to model the performance of the MATE algorithm, one first needs to understand the computational and communication requirements of each of the MATE's tasks, taking into perspective the type of problems intended to be solved.

For the processes performing tasks associated with subsystem $\mathbb{S}_k$, their execution time $T(\mathbb{S}_k)$ is defined as follows:

$$T(\mathbb{S}_k) = T^{Thv}_{comp}(\mathbb{S}_k) + T^{Thv}_{comm}(\mathbb{S}_k) + T^{lnk}_{idle}(\mathbb{S}_k) + T^{i}_{comm}(\mathbb{S}_k) + T^{v}_{comp}(\mathbb{S}_k) \tag{3.1}$$

where,

$T^{Thv}_{comp}(\mathbb{S}_k)$ = time spent computing the multi-node Thévenin equivalent $\mathbf{Z}^b_k$ and $\mathbf{e}^b_k$ for the subsystem $\mathbb{S}_k \in \mathbb{S}$;
$T^{Thv}_{comm}(\mathbb{S}_k)$ = time spent sending $\mathbf{Z}^b_k$ and $\mathbf{e}^b_k$ to the link solver process;
$T^{lnk}_{idle}(\mathbb{S}_k)$ = time spent waiting for the link solver process to compute the link currents $\mathbf{i}^l$;
$T^{i}_{comm}(\mathbb{S}_k)$ = time spent receiving the border nodes injections $\mathbf{i}^b_k$ from the link solver;
$T^{v}_{comp}(\mathbb{S}_k)$ = time spent solving the node voltages $\mathbf{v}_k$ for the updated current injections $\mathbf{i}_k$.

For the link solver process, which solves equations associated with the link system $\mathbb{L}$, or multi-area Thévenin equivalent, the execution time is given by:

$$T(\mathbb{L}) = T^{subs}_{idle}(\mathbb{L}) + T^{Thv}_{comm}(\mathbb{L}) + T^{i}_{comp}(\mathbb{L}) + T^{i}_{comm}(\mathbb{L}) \tag{3.2}$$

where,

$T^{subs}_{idle}(\mathbb{L})$ = time spent waiting for the subsystems $\mathbb{S}_k \in \mathbb{S}$ to compute $\mathbf{Z}^b_k$, $\mathbf{e}^b_k$ and $\mathbf{v}_k$;
$T^{Thv}_{comm}(\mathbb{L})$ = time spent receiving $\mathbf{Z}^b_k$ and $\mathbf{e}^b_k$ from the subsystems $\mathbb{S}_k \in \mathbb{S}$;

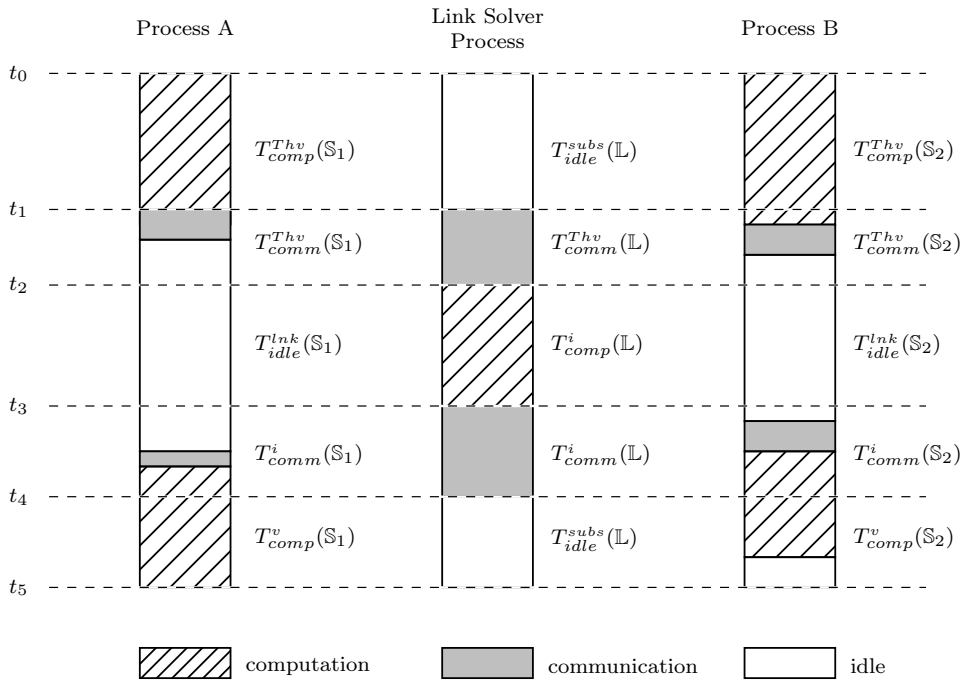


Figure 3.2. MATE timeline for two subsystems, $\mathbb{S}_1$ and $\mathbb{S}_2$, and the link solver.

$T^{i}_{comp}(\mathbb{L})$ = time spent setting up and computing the link current equations;
$T^{i}_{comm}(\mathbb{L})$ = time spent scattering the border node current injections $\mathbf{i}^{b}_{k}$ to the subsystems $\mathbb{S}_k \in \mathbb{S}$.

All of the above timings are shown in the MATE algorithm timeline example, illustrated in the Figure 3.2. Three processes are shown: two subsystem processes A and B, and the link solver process. At instant t_0, all subsystem processes start computing their corresponding multi-node Thévenin equivalents. Once the first of them is ready, at instant t_1, the link solver process can start receiving it, while the other processes finish their computation. Right after finishing sending their multi-node Thévenin equivalents to the link solver, the subsystems start waiting for the link solver feedback. At instant t_2, the link solver received all Thévenin equivalents and starts setting up and computing the link current equations. Once the link currents become available, the link solver process sends them to the subsystems at instant t_3. As soon as the subsystems receive the border nodes current injection due to their local links, they are able to update their local current injections and solve their local node voltages. After instant t_4, the link solver process remains waiting for the next batch of Thévenin equivalents, while the subsystems finish updating the local variables.

From the previous discussion on the MATE timeline, the total time for the network-based MATE algorithm, T_{MATE}, can be estimated by the sum of the terms related to the most time-consuming subsystem $\mathbb{S}_{max}$, in addition to the data exchanges and computational tasks

associated with the link currents. Thus, the foregoing estimation can be denoted as follows:

$$T_{MATE} = T^{Thv}_{comp}(\mathbb{S}_{max}) + T^{Thv}_{comm}(\mathbb{L}) + T^{i}_{comp}(\mathbb{L}) + T^{i}_{comm}(\mathbb{L}) + T^{v}_{comp}(\mathbb{S}_{max}) \tag{3.3}$$

Next, each time component, introduced above, will be formulated, based on the computational complexity or communication requirements of each of the MATE's tasks described in Section 2.3. In summary, the computational and communication tasks covered in the following sections are:

- Computational Tasks:
  - Multi-node Thévenin equivalents computation
  - Link currents computation
  - Subsystems update
- Communication Tasks:
  - Subsystems multi-node Thévenin equivalents gathering
  - Link currents scattering

3.2.2 Computational Aspects of MATE

In general, the most computationally expensive tasks of the MATE algorithm are the computation of the multi-node Thévenin equivalents at the subsystems level, and the solution of the link currents at the link solver level.

Since the main purpose of the MATE algorithm is to tackle very large systems, the subsystems, obtained by means of any partitioning heuristic, will be also large enough to render these subsystems very sparse. Therefore, the application of well-known sparsity techniques will be fundamental to guarantee efficiency of the subsystems tasks. As for the link solver tasks, traditional dense matrix operations are required.

Multi-node Thévenin equivalents

Based on (2.12) and (2.16), notice that both $\mathbf{e}^b_k$ and $\mathbf{Z}^b_k$ are solutions of the same linear system (2.1) defined by $\mathbf{Y}_k$, with only different right-hand sides, i.e., $\mathbf{j}_k$ for computing $\mathbf{e}^b_k$ and $(\mathbf{Q}_k)^T$ for computing $\mathbf{Z}^b_k$. In this way, each subsystem's local network, expressed by (2.1) has to be solved b_k times for building $\mathbf{Z}^b_k$, and one time for $\mathbf{e}^b_k$. Since the matrix $\mathbf{Y}_k$ is usually very sparse, sparse techniques for solving this linear system should be employed. A widely used technique for solving large sparse linear systems is *LU factorization*.

The time necessary to compute multi-node Thévenin equivalents can be then approximated by the following expression:

$$T_{comp}^{Thv}(\mathbb{S}_k) = T_{1fact}(\mathbb{S}_k) + (N_z - 1)\, T_{2fact}(\mathbb{S}_k) + (N_z\, b_k + N_i)\, T_{solv}(\mathbb{S}_k) \tag{3.4}$$

where $T_{1fact}(\mathbb{S}_k)$ represents the time for the first factorization of the admittance matrix $\mathbf{Y}_k$, which includes both symbolic and numerical factorizations. In turn, $T_{2fact}(\mathbb{S}_k)$ is the time spent in the numerical factorization that reuses the structures of the factors $\mathbf{L}_k$ and $\mathbf{U}_k$, acquired during the symbolic factorization of $\mathbf{Y}_k$. Additionally, $T_{solv}(\mathbb{S}_k)$ is the time for solving the triangular systems, defined by the factors $\mathbf{L}_k$ and $\mathbf{U}_k$. The parameter N_z captures the number of topological changes in subsystem $\mathbb{S}_k$, i.e., how often $\mathbf{Z}_k^b$ needs to be recalculated, whereas the parameter N_i represents the number of changes in the injected currents.

Results obtained by Alvarado (1976) with respect to the order of complexity of sparse factorization and repeated solutions of triangular systems suggests that $T_{1fact}(\mathbb{S}_k)$, $T_{2fact}(\mathbb{S}_k)$ and $T_{solv}(\mathbb{S}_k)$ can be approximated by exponential functions, as follows:

$$T_{1fact}(\mathbb{S}_k) = a_3\, {\rho_k}^2\, n_k + a_2\, \rho_k\, n_k + a_1\, n_k + a_0 \tag{3.5}$$

$$T_{2fact}(\mathbb{S}_k) = b_3\, {\rho_k}^2\, n_k + b_2\, \rho_k\, n_k + b_1\, n_k + b_0 \tag{3.6}$$

$$T_{solv}(\mathbb{S}_k) = c_2\, \rho_k\, n_k + c_1\, n_k + c_0 \tag{3.7}$$

where n_k is the number of buses in subsystem $\mathbb{S}_k$ and, ρ_k represents the branch-to-bus ratio of the same subsystem. These relationships show that more interconnected systems, i.e., higher branch-to-bus ratios ρ_k, are more computationally expensive. On one hand, if a system has all its buses interconnected to one another, its branch-to-ratio ρ equals $n - 1$, which makes its factorization and repeated solution time of order $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively. On the other hand, for loosely interconnected systems, as is the case of most power systems, where each bus connects to just a few other neighboring buses, the factorization and repeated solution times vary practically linearly with the number of buses n in the system, which means an order of complexity $\mathcal{O}(n)$[6].

In summary, the aspects that influence the computation of the multi-node Thévenin equivalents are: subsystem size, topology, ordering of the nodes, as well as the number of border nodes. Both size and number of border nodes are highly dependent on the partitioning used to tear the original system apart. Ideally, the employed partitioning technique should make subsystems as balanced as possible, while keeping the number of border nodes

[6]For further details, see Appendix A

reasonably small. This also helps in reducing the amount of data that needs to be exchanged with the link solver.

Subsystems update

Once the local current injections $\mathbf{j}_k + \mathbf{i}_k$ are available, the sparse linear system (2.2) needs to be solved. Since the subsystem matrix $\mathbf{Y}_k$ is the same as the one used to compute the multi-node Thévenin equivalents, the factors $\mathbf{L}_k$ and $\mathbf{U}_k$ can be reused to solve (2.2) for the node voltages $\mathbf{v}_k$, by means of sparse forward/backward substitutions. Hence, using $T_{solv}(\mathbb{S}_k)$, which is defined in (3.7), the time required to calculate $\mathbf{v}_k$ is as follows:

$$T^{v}_{comp}(\mathbb{S}_k) = N_i \, T_{solv}(\mathbb{S}_k) \tag{3.8}$$

Notice from (3.8) that each subsystem has to be solved as many times as their current injections change, i.e., N_i times.

Link currents computation

Considering that all p subsystems multi-node Thévenin equivalents are available at the link solver, one can build the multi-area Thévenin equivalent, formed by $\mathbf{Z}^l$ and $\mathbf{e}^l$, by means of the relationships denoted in (2.25). In these equations, the link Thévenin equivalents, defined by (2.22), are built in-place on top of the multi-area Thévenin equivalent, $\mathbf{Z}^l$ and $\mathbf{e}^l$, according to (3.9).

$$\mathbf{Z}^l = \mathbf{Z}^l_0 + \sum_{k=1}^{p} \left(\mathbf{R}_k\right)^T \mathbf{Z}^b_k \mathbf{R}_k \tag{3.9a}$$

$$\mathbf{e}^l = \sum_{k=1}^{p} \left(\mathbf{R}_k\right)^T \mathbf{e}^b_k \tag{3.9b}$$

Once the multi-area Thévenin equivalent seen from the links is built, the link currents computation can proceed. In this case, the dense linear system denoted in (2.25c) needs to be solved, by first factorizing $\mathbf{Z}^l$ and then solving the triangular systems associated with the obtained dense $\mathbf{L}$ and $\mathbf{U}$ factors, by means of forward and backward substitutions. Moreover, since subsystems may suffer topological changes during simulations, their correspondent link Thévenin equivalents $\mathbf{Z}^l_k$ will also change. Therefore, if any subsystem $\mathbb{S}_k \in \mathbb{S}$ has its topology changed N_z times, the matrix $\mathbf{Z}^l$ also needs to be re-factorized N_z times. Similarly, if subsystems current injections change N_i times over a simulation period, the link Thévenin voltages $\mathbf{e}^l$ also have to be computed N_i times. So, based on the fact that the number of

operations required to factorize $\mathbf{Z}^l$ is of the order $\mathcal{O}(l^3)$, and forward/backward substitutions of the order $\mathcal{O}(l^2)$, one can conclude that:

$$T^i_{comp}(\mathbb{L}) = N_z\, T_{fact}(\mathbb{L}) + N_i\, T_{solv}(\mathbb{L}) \tag{3.10}$$

where, $T_{fact}(\mathbb{L})$ and $T_{solv}(\mathbb{L})$ can be approximated by

$$T_{fact}(\mathbb{L}) = a_3\, l^3 + a_2\, l^2 + a_1\, l \tag{3.11}$$

$$T_{solv}(\mathbb{L}) = b_2\, l^2 + b_1\, l \tag{3.12}$$

3.2.3 Communication Aspects of MATE

In the MATE algorithm, two main communication tasks are necessary: gathering the multi-node Thévenin equivalents in the link solver and scattering link currents to subsystems.

For establishing the performance model of such routines the PLogP model, introduced by Kielmann et al. (2000), was employed. Known as the *parameterized LogP* model, PLogP is an extension of the LogP model, originally proposed by Culler et al. (1996). Differently from the original LogP, which considers fixed parameters for modeling performance of dedicated networks, the PLogP includes the dependencies on the messages sizes. The parameters which both PLogP employs are described as follows:

- L: end-to-end latency from process to process, which includes the time for copying the data to and from the network interfaces and transferring the data over the physical network;
- $o_s(m)$: period of time in which the processor is engaged in sending the message of size m;
- $o_r(m)$: period of time in which the processor is engaged in receiving the message of size m;
- $g(m)$: minimum time, or gap, between consecutive message transmissions or receptions, which also considers all contributing factors, including $o_s(m)$ and $o_r(m)$.

In the next sections, the link currents scattering will be first explained, followed by the multi-node Thévenin equivalents gathering.

Link currents scatter

At this stage of the solution, each of the p subsystems $\mathbb{S}_k \in \mathbb{S}$ has to receive from the link solver process, $\mathbb{L}$, the net currents $\mathbf{i}^b_k$ injected by the l_k local links at the b_k border nodes.

These current injections are the result of a linear combination of the link currents $\mathbf{i}^l$ obtained by means of the link-to-border mapping $\mathbf{R}_k$, as shown in (3.13).

$$\mathbf{i}_k^b = \mathbf{R}_k \, \mathbf{i}^l \tag{3.13}$$

The procedure described by (3.13) represents, in fact, a pre-processing step performed by the link solver on the link currents $\mathbf{i}^l$ just before sending them to the subsystems. As mentioned earlier, by sending the border nodes net current injections $\mathbf{i}_k^b$, instead of the global link currents vector $\mathbf{i}^l$, one reduces the communication overhead of the algorithm.

In addition, once the subsystems have received their net injections $\mathbf{i}_k^b$, they are added to the local subsystem current injection $\mathbf{j}_k$ by means of the border-to-subsystem mapping $(\mathbf{Q}_k)^T$ as shown in (3.14). The final vector $\mathbf{j}_k$ is later used for the subsystem update, as discussed in Section 3.2.2.

$$\mathbf{j}_k \leftarrow \mathbf{j}_k + (\mathbf{Q}_k)^T \, \mathbf{i}_k^b \tag{3.14}$$

In fact, the present communication task corresponds to a customized scatter function, according to the MPI Standard (2008). This is shown graphically in Figure 3.3. In this procedure, parameters L, $o_s(m)$, $o_r(m)$ and $g(m)$ characterize the behavior of a network, according to the PLogP model (Kielmann et al., 2000).

The link currents scatter routine starts in the link solver process, which sends p consecutive messages to distinct processes handling the subsystems. As seen in Figure 3.3, the first message alone takes a time equal to $o_s(m_1) + L + o_r(m_1)$ to be completely received by the process. However, before the link solver can start sending the second message, it demands a minimum of $g(m_1)$ seconds before the message can be sent. The second message leaves the link solver process only at the $o_s(m_2) + g(m_1)$ mark and is completely received in the respective subsystems by $g(m_1) + o_s(m_2) + L + o_r(m_2)$. Following the previous reasoning, one can find the time consumed by the link solver process to send p messages of size m_k $(k = 1, \ldots, p)$ to the subsystems processes. This time is given by $L + o_s(m_p) + o_r(m_p) + \sum_{k=1}^{p-1} g(m_k)$. Finally the link currents scatter timing $T_{comm}^i(L)$ is given in (3.15).

$$T_{comm}^i(\mathbb{L}) = L + o_s(m_p) + o_r(m_p) + \sum_{k=1}^{p-1} g(m_k) \tag{3.15}$$

Recalling that each message of size m_k is associated with currents being injected at the b_k border nodes of subsystem $\mathbb{S}_k$, there is a linear relationship between m_k and b_k. In fact, this relationship depends on the data type that represents each injection. If a single current

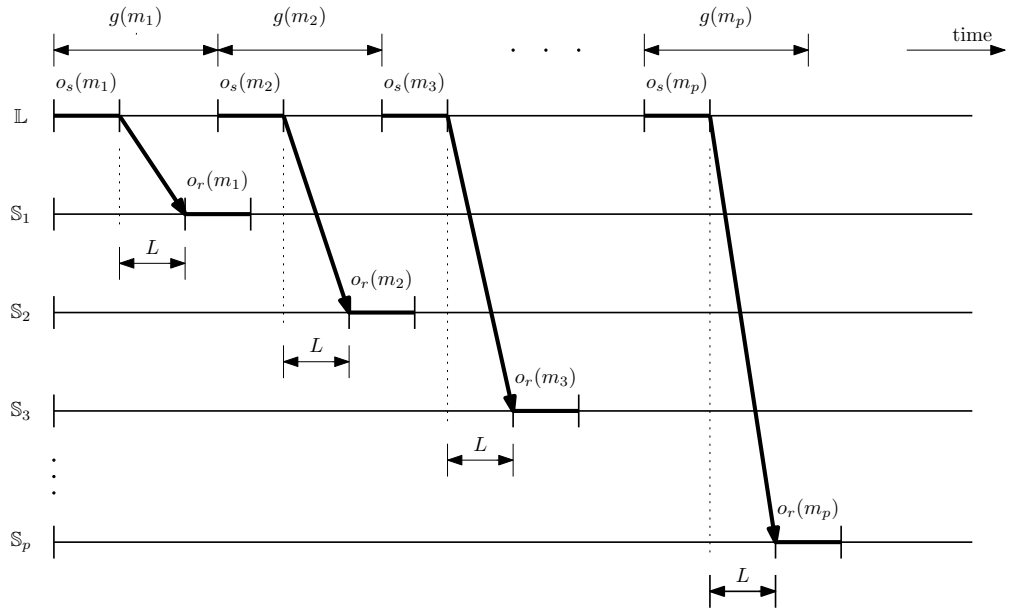


Figure 3.3. Link currents scatter according to PLogP model.

injection takes c bytes of memory, the message size m_k is defined by (3.16).

$$m_k = c\, b_k \tag{3.16}$$

Considering that longer messages take longer periods of times to be transfered to another process, one can conclude that the function $g(m_k)$ is always positive and increases monotonically with the message size m_k. Therefore, for any message size m_k, $g(m_k) \leq g(\overline{m}_k)$, where $\overline{m}_k$ is the longest message amongst all m_k, for $k = 1, \ldots, p$. Consequently, the following inequality is also true.

$$\sum_{k=1}^{p-1} g(m_k) \leq \sum_{k=1}^{p-1} g(\overline{m}_k) = (p-1)\, g(\overline{m}_k) \tag{3.17}$$

That fact that the gap g includes both overheads o_s and o_r leads to $g(m_k) \geq o_s(m_k)$ and $g(m_k) \geq o_r(m_k)$ (Kielmann et al., 2000). Combining these last pieces of information with (3.15) and (3.17) yields (3.18), which represents an upper bound for the link currents scatter time $T^i_{comm}(\mathbb{L})$.

$$T^i_{comm}(\mathbb{L}) \leq L + (p+1)\, g(\overline{m}_k) \tag{3.18}$$

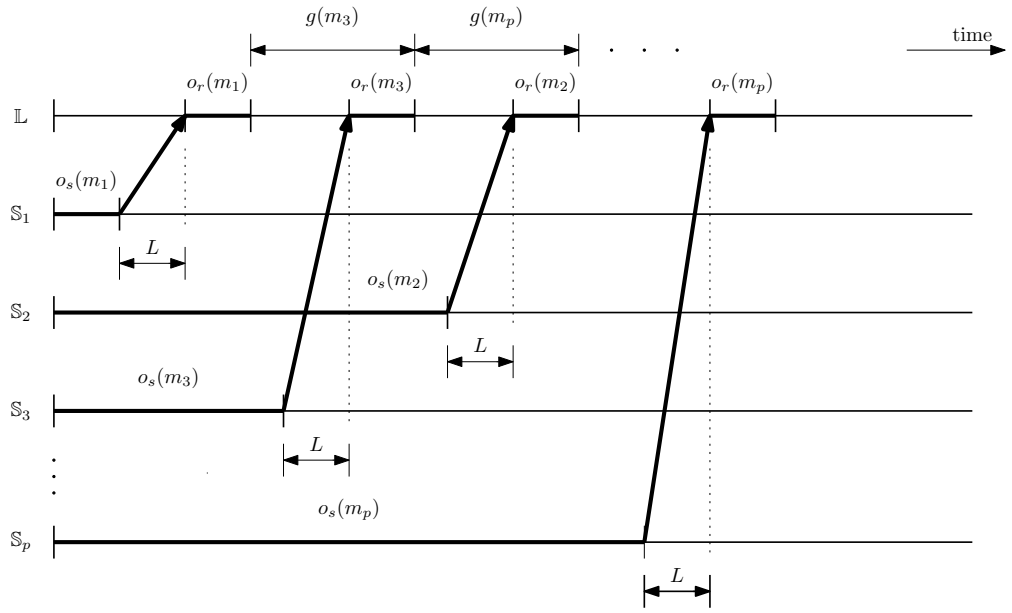


Figure 3.4. Multi-node Thévenin equivalents gather according to PLogP model.

Multi-node Thévenin equivalents gather

In this case, each of the p subsystems $\mathbb{S}_k \in \mathbb{S}$ has to send its previously computed multi-node Thévenin equivalent, formed by $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$, to the link solver process. In fact, this task corresponds to a customized gather function (MPI Standard, 2008).

Considering that each one of the multi-node Thévenin equivalents is associated with a message of size m_k, the gather routines follows the procedure described in Figure 3.4. In light of the PLogP model (Kielmann et al., 2000), the time that the link solver takes to receive p consecutive messages of size m_k corresponds to $L + o_s(m_p) + o_r(m_p) + \sum_{k=1}^{p-1} g(m_k)$ seconds.

Differently from the current links scatter, for the multi-node Thévenin gather procedure two kinds of messages need to be considered: full multi-node Thévenin equivalents, consisting of $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$ and, the multi-node Thévenin equivalent voltage $\mathbf{e}_k^b$ alone. While the first kind will be sent to the link solver only when a subsystem topology changes, the second one is sent whenever the internal current injections vary. The size of the messages carrying the information of the multi-node Thévenin equivalents of the subsystem $\mathbb{S}_k$ are summarized by (3.19).

$$m_k = \begin{cases} c\, b_k\, (b_k + 1) & \text{if } \mathbf{Z}_k^b \text{ and } \mathbf{e}_k^b \text{ are sent,} \\ c\, b_k & \text{if only } \mathbf{e}_k^b \text{ is sent.} \end{cases} \tag{3.19}$$

where c represents the amount of memory, in bytes, that each element of either $\mathbf{Z}_k^b$ or $\mathbf{e}_k^b$ takes.

Finally, the communication time $T_{comm}^{Thv}(\mathbb{L})$, perceived by the link solver process, can be calculated as follows:

$$T_{comm}^{Thv}(\mathbb{L}) = L + o_s(m_p) + o_r(m_p) + \sum_{k=1}^{p-1} g(m_k) \tag{3.20}$$

Notice that when only $\mathbf{e}_k^b$ needs to be gathered in the link solver, $T_{comm}^{Thv}(\mathbb{L})$ is equal to the link currents scatter time $T_{comm}^{i}(\mathbb{L})$, defined in (3.15). Moreover, the upper bound (3.18) found for $T_{comm}^{i}(L)$ is applicable to $T_{comm}^{Thv}(\mathbb{L})$ as well, which leads to (3.21).

$$T_{comm}^{Thv}(\mathbb{L}) \leq L + (p+1)\, g(\overline{m}_k) \tag{3.21}$$

3.2.4 MATE Speedup and Efficiency

According to (Foster, 1995), both speedup and efficiency are relative quantities, which are measured, usually, with respect to the best known algorithm available to perform the same task under analysis. Hence, since sparsity-based network solvers are widely used in power system industry (Tinney & Walker, 1967), they appear a natural choice as a comparative baseline to estimate speedups and efficiency.

Sparsity-based solver computation time

When applying sparsity techniques alone to solve a large electric network, two basic tasks need to be performed: firstly, factorization of the system matrix $\mathbf{Y}$ and, secondly, solution of two triangular sparse systems for each current injection $\mathbf{i}$. Thus, keeping in mind that the system topology may change N_z times and its current injections changes N_i times, the sparsity-based network solver computation time T_{SPARSE} may be defined as denoted bellow:

$$T_{SPARSE} = T_{1fact}(\mathbb{S}) + (N_z - 1)\, T_{2fact}(\mathbb{S}) + N_i\, T_{solv}(\mathbb{S}) \tag{3.22}$$

where $T_{1fact}(\mathbb{S})$, $T_{2fact}(\mathbb{S})$ and $T_{solv}(\mathbb{S})$ follow the relationships (3.5), (3.6) and (3.7), respectively.

MATE versus Sparsity Alone

Plugging (3.3) and (3.22) into the definitions for speedup and efficiency yields the following expressions:

$$\mathcal{S}_{MATE} = \frac{T_{SPARSE}}{T_{MATE}} \tag{3.23}$$

$$\mathcal{E}_{MATE} = \frac{\mathcal{S}_{MATE}}{p+1} \tag{3.24}$$

The above expressions synthesize multivariable functions dependent on aspects which involve characteristics of the system to be solved, such as system topology and partitioning strategy[7]. Furthermore, software implementation and hardware capabilities also considerably impacts metrics, such as speedup and efficiency. Hence, the performance analysis of the MATE algorithm needs to be evaluated on a case by case basis, which needs to consider the system under study, in addition to the computer architecture and software available. In the following section, the proposed MATE performance model will be evaluated for a test case recently presented in (Tomim et al., 2008).

3.2.5 MATE Performance Qualitative Analysis

In order to provide means to qualitatively understand the constraints in the network-based MATE algorithm, a qualitative analysis based on the computational and communication costs, discussed previously, will be presented.

For this analysis, one first splits the total MATE timing T_{MATE}, given by (3.3), into three distinct components, given by (3.25): T_P, T_L and T_C. The component T_P is associated with parallel tasks performed by the processes handling subsystems. The second component T_L refers to sequential computations carried out by the link solver. Finally, the component T_C accumulates the communication overhead incurred by the MATE algorithm.

$$T_P = T^{Thv}_{comp}(\mathbb{S}_{max}) + T^{v}_{comp}(\mathbb{S}_{max}) \tag{3.25a}$$

$$T_L = T^{i}_{comp}(\mathbb{L}) \tag{3.25b}$$

$$T_C = T^{Thv}_{comm}(\mathbb{L}) + T^{i}_{comm}(\mathbb{L}) \tag{3.25c}$$

Employing the expressions obtained in Section 3.2, the set of timings above can be explicitly written in terms of the variables that directly affect the performance of the MATE

[7]Note that p refers to the number of partitions a given system is torn into, and the number of processors employed in the MATE algorithm equals $p+1$. Therefore, $p+1$ processors is used in order to compute the efficiency $\mathcal{E}_{MATE}$.

algorithm, as shown in (3.26).

$$T_P = \big[N_z\, k_{sf} + (N_z\, b_{max} + 2\, N_i)k_{ss}\big]\, n_{max} \tag{3.26a}$$

$$T_L = N_z\, k_{df}\, l^3 + N_i\, k_{ds}\, l^2 \tag{3.26b}$$

$$T_C = (p+1)\big[2\, N_i\, g(m_e) + N_z\, g(m_t)\big] \tag{3.26c}$$

where m_e and m_t are relative to the memory associated with the Thévenin equivalent voltages and the full Thévenin equivalent, respectively. Furthermore, n_{max} represents the order of the biggest subsystem $\mathbb{S}_{max}$, which, for well balanced partitions, can be approximated by $\frac{N}{p}$, where p the number of subsystems that form the original untorn system of order N.

Under similar circumstances, the sequential sparse solver timing T_{SPARSE}, defined in (3.22), assumes the following form:

$$T_{SPARSE} = (N_z\, k_{sf} + N_i\, k_{ss})\, N \tag{3.27}$$

Plugging the expressions defined above into the speedup definition (3.23) gives

$$\mathcal{S}_{MATE} \approx \frac{p}{\frac{N_z\, k_{sf} + (N_z\, b_{max} + 2\, N_i)k_{ss}}{N_z\, k_{sf} + N_i\, k_{ss}} + \frac{N_z\, k_{df}\, l^3 + N_i\, k_{ds}\, l^2}{\big(N_z\, k_{sf} + N_i\, k_{ss}\big)n_{max}} + \frac{(p+1)[2\, N_i\, g(m_e) + N_z\, g(m_t)]}{\big(N_z\, k_{sf} + N_i\, k_{ss}\big)n_{max}}} \tag{3.28}$$

Next, two separate analyzes on the speedup $\mathcal{S}_{MATE}$, introduced in (3.28) will be discussed: (a) for when the number of factorizations N_z approximates the number of repeated solutions N_i, and (b) for when the number of repeated N_i is much greater than the number of factorizations N_z.

Case for $N_z \approx N_i$

In such a situation, (3.28) becomes:

$$\mathcal{S}_{MATE}\Big|_{N_z \approx N_i} \approx \frac{p}{\frac{k_{sf} + (b_{max} + 2)k_{ss}}{k_{sf} + k_{ss}} + \frac{k_{df}\, l^3 + k_{ds}\, l^2}{\big(k_{sf} + k_{ss}\big)n_{max}} + \frac{(p+1)[2\, g(m_e) + g(m_t)]}{\big(k_{sf} + k_{ss}\big)n_{max}}} \tag{3.29}$$

In general, subsystems become more interconnected as the number of partitions increase, which makes the number of border nodes b_{max}, number of links l, and the communication-related $g(m_e)$ and $g(m_t)$ increase as well, while only n_{max} decreases. Therefore, the denominator of (3.29) will always increase. Moreover, the term associated with the number of links l can be expected to present the faster increase ratio, due to its cubic power.

As a result, the network-based MATE algorithm will drastically loose performance when-

ever systems require frequent factorizations. Such performance loss rate can only be alleviated in case the hardware/software-associated coefficients k_{df}, k_{ds}, $g(m_e)$ and $g(m_t)$ can be reduced. However, such reduction can only be achieved up to some extent, by means of specialized hardware/software for dense matrix operations and data communication.

Case for $N_z \ll N_i$

In this case, the speedup achieved with the MATE algorithm approximates the expression denoted in (3.30).

$$\mathcal{S}_{MATE}\Big|_{N_z \ll N_i} \approx \frac{p}{2 + \frac{k_{ds}\, l^2}{k_{ss}\, n_{max}} + \frac{2(p+1)g(m_e)}{k_{ss}\, n_{max}}} \tag{3.30}$$

Following the trend of the previous case, the denominator in the speedup expression still tends to increase as the number of partitions grows. However, its increase occurs at a much slower rate, due to the quadratic term associated with the number of links l.

From the expression above, even for partitions that are very weakly connected, i.e., $l \ll n_{max}$, the speedup would be still limited by $\frac{p}{2}$, which represents the theoretical speedup limit, in case both dense operations in the link solver and data communications were instantaneous. This is explained by the fact that each subsystem is solved twice, each time the original untorn system is solved.

In order to keep the speedup as close to $\frac{p}{2}$ as possible, the two varying denominator terms should be much less than the unity. In this sense, these terms can be seen as penalty factors on the network-based MATE algorithm due to the sequential computations performed on the link solver and the data exchange among processes. These penalty factors are defined in (3.31), which also states the constraints that should be imposed on the same penalty factors in order to maximize the algorithm's performance.

$$\lambda_{link} = \frac{k_{ds}\, l^2}{k_{ss}\, n_{max}} \ll 1 \tag{3.31a}$$

$$\lambda_{comm} = \frac{2\,(p+1)\, g(m_e)}{k_{ss}\, n_{max}} \ll 1 \tag{3.31b}$$

The inequality given in (3.31a) serves as a constraint on the number of links that a system of size N can have, whenever it is torn into p subsystems. In other words, it reflects the fact that the workload in the subsystems has to be much heavier than the workload in the link solver. Therefore, the MATE algorithm has better chances to achieve higher speedup when solving larger systems. Constraint (3.31b) provides a bound on the communication time associated with Thévenin equivalent voltages, which depend on the intercommunication gap $g(m_e)$. Similarly to the previous term, it reflects the fact that the Thévenin equivalents

communication overhead should be much lower than the computations. In other words, subsystems cannot be too small, otherwise the Thévenin equivalents communication timings would eventually surpass their computation time, degrading considerably the algorithm's performance.

3.3 Hardware/Software Benchmarks

The parameters that describe the performance of the hardware and software combination available is of fundamental importance to predict the performance of the network-based MATE implementation without actually implementing it.

Sparse and dense operations modules as well as interprocess communication kernels were benchmarked for the distributed computing environment available in the UBC Power Engineering Group Laboratory. As detailed in (De Rybel et al., 2008), this environment consists of 16 AMD Athlon^{TM} 64 2.5 GHz PC units built on a single rack, interconnected by two distinct private networks: one built with Dolphin SCI (Scalable Coherent Interface) network cards and another built with Gigabit Ethernet cards.

3.3.1 Sparse Linear Solver Benchmark

For the sparse routines, two specific kernels need to be benchmarked, namely, the LU factorization and repeated triangular solutions.

In this study the sparse linear solver kernels implemented in SuperLU 3.0 library were employed (Demmel et al., 1999; Li et al., 2003). For the timing procedure, a number of WECC subsystems, generated in the partitioning stage (see Section 3.4.1, page 70), were fully solved, i.e., factorized and solved, 1000 times. Computation time averages are plotted in Figure 3.5.

Another important aspect that has be considered when solving large sparse linear systems is minimizing fill-in of the factors **L** and **U** during the factorization process. Minimizing the fill-in has a twofold benefit: minimizing the memory consumption and the number of required floating-point operations. Thus, choosing a good ordering technique for the sparse matrices of interest ensures efficiency of their solutions. For the present work, the employed ordering technique is known as multilevel nested dissection, whose implementation is available in the METIS 4.0 library (Karypis & Kumar, 1998a,b), introduced in Section 3.4.1.

By means of linear regression, the timing data previously acquired were fitted in the modeling functions (3.5), (3.6) and (3.7). A summary of the results of the data fitting procedure is presented in Table 3.2. In this table, the parameters associated with each component of

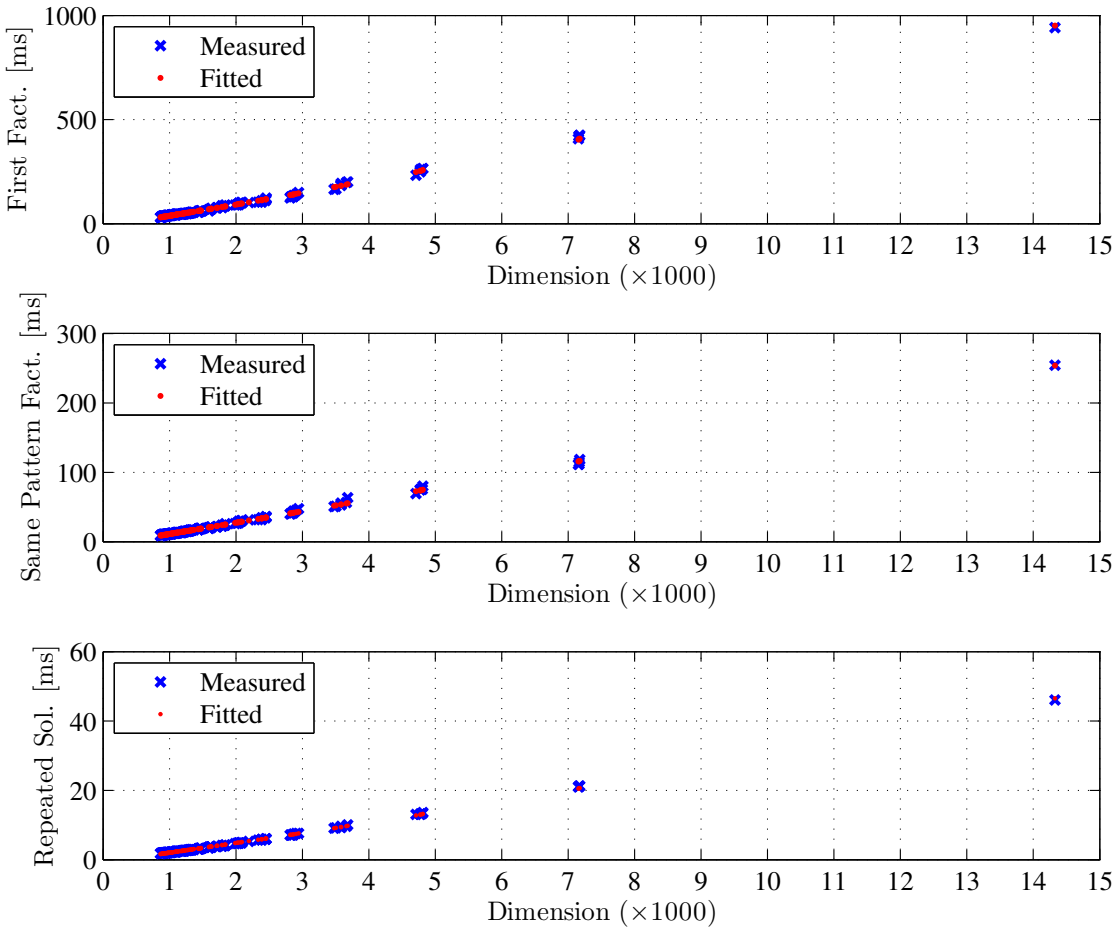


Figure 3.5. Sparse operations benchmark.

the modeling functions are shown. The term associated with $\rho^2 n$ is not observable in the fitted function, which shows that, the computation effort, required by the SuperLU routines to solve the WECC system, increases linearly with the size of the subsystems. This is the same behavior expected from sparse band matrices (Alvarado, 1976).

The sparse routines throughputs are given in Figure 3.7. The results show that the repeated solutions are more efficient than the factorizations in terms of floating-point operations per second. As for the factorizations, it can be observed that the symbolic factorization required by the first-time factorization reduces considerably the efficiency of the routine when compared to the same-pattern factorization.

Table 3.1. Data fitting summary for the sparse operations.

First Factorization ($R^2 = 0.99749$)		
Component	Value	C.I. (95%)*
$\rho\, n$	1.4956×10^{-11}	$\pm 9.7840 \times 10^{-13}$
n	5.0778×10^{-6}	$\pm 8.1457 \times 10^{-8}$
1	-1.3855×10^{-3}	$\pm 1.1733 \times 10^{-4}$
Same Pattern Factorization ($R^2 = 0.99803$)		
Component	Value	C.I. (95%)*
$\rho\, n$	2.0521×10^{-12}	$\pm 2.4136 \times 10^{-13}$
n	1.5777×10^{-6}	$\pm 2.0094 \times 10^{-8}$
1	-4.7687×10^{-4}	$\pm 2.8943 \times 10^{-5}$
Repeated Solution ($R^2 = 0.99869$)		
Component	Value	C.I. (95%)*
$\rho\, n$	5.6283×10^{-13}	$\pm 3.5275 \times 10^{-14}$
n	2.6740×10^{-7}	$\pm 2.9368 \times 10^{-9}$
1	-6.7353×10^{-5}	$\pm 4.2301 \times 10^{-6}$

* 95% confidence interval for the parameters estimates.

3.3.2 Dense Linear Solver Benchmark

Following the same procedure adopted for timing the sparse solver kernels, both dense routines, namely, the LU factorization and the repeated triangular solutions, were benchmarked. The dense routines employed in this study are implemented in the GotoBLAS library (Goto, 2006; Goto & Van De Geijn, 2008a,b), which provides a large set of highly optimized BLAS (*Basic Linear Algebra Subprograms*) (Blackford et al., 2002) and LAPACK (*Linear Algebra Package*) (Anderson et al., 1999) routines.

For the benchmarking process, instead of using matrices extracted from the previous partitions (multi-node Thévenin equivalents, for instance), randomly generated dense matrices were fully solved 1000 times each. The measured timings are shown in Figure 3.6. The results of the data fitting procedure, using the functions defined in (3.11) and (3.12), are shown Table 3.1.

The throughput for the dense factorization and repeated solution is shown in Figure 3.8. Differently from what was observed for the sparse routines, the factorization is more efficient that the repeated solutions in terms of floating-point operations per second. This can be explained by the fact that, for dense matrices, the factorizations can be performed in blocks. Such characteristic allows implementations in modern processors, such as GotoBLAS, that

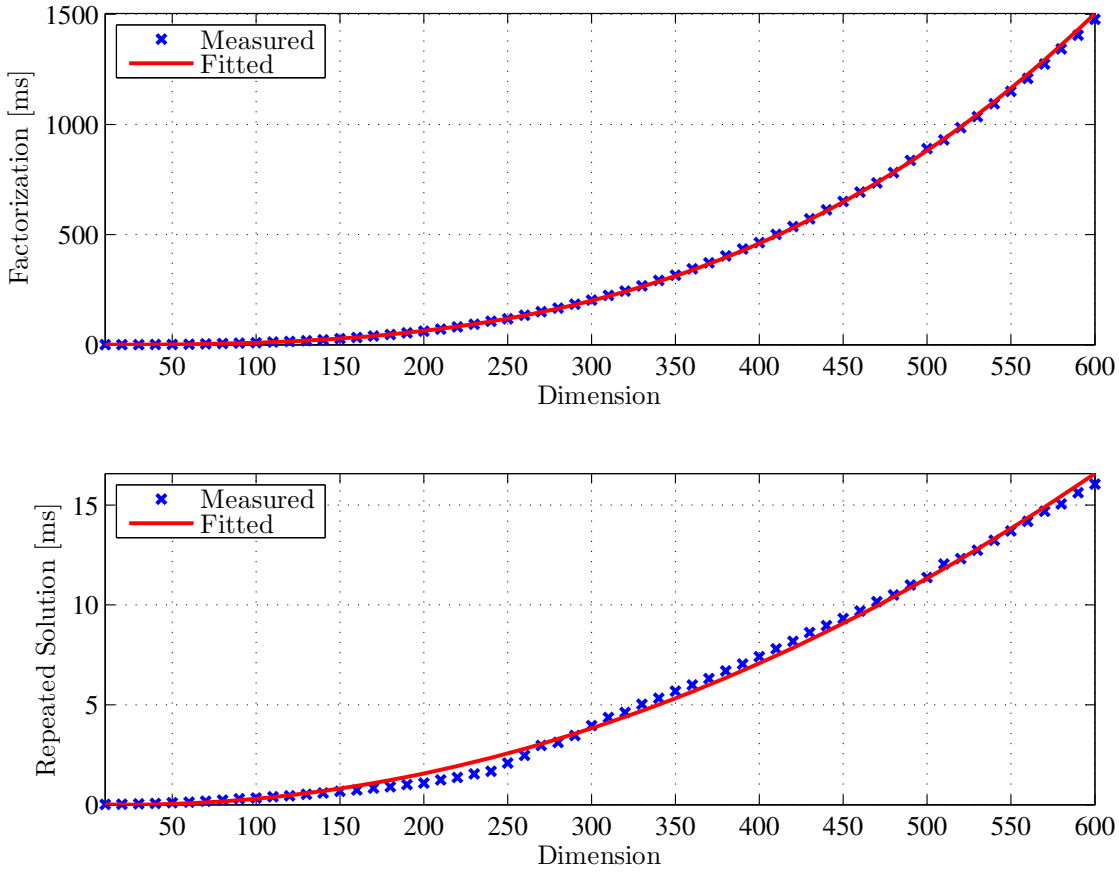


Figure 3.6. Dense operations benchmark.

minimize considerably RAM memory access and maximize cache memory usage.

Such an improved performance of dense matrix operations is often exploited in sparse-oriented algorithms. For instance, blocked strategies for handling sparse matrices are used in the aforementioned SuperLU (Demmel et al., 1999; Li et al., 2003) and UMFPACK (Davis, 2004) libraries.

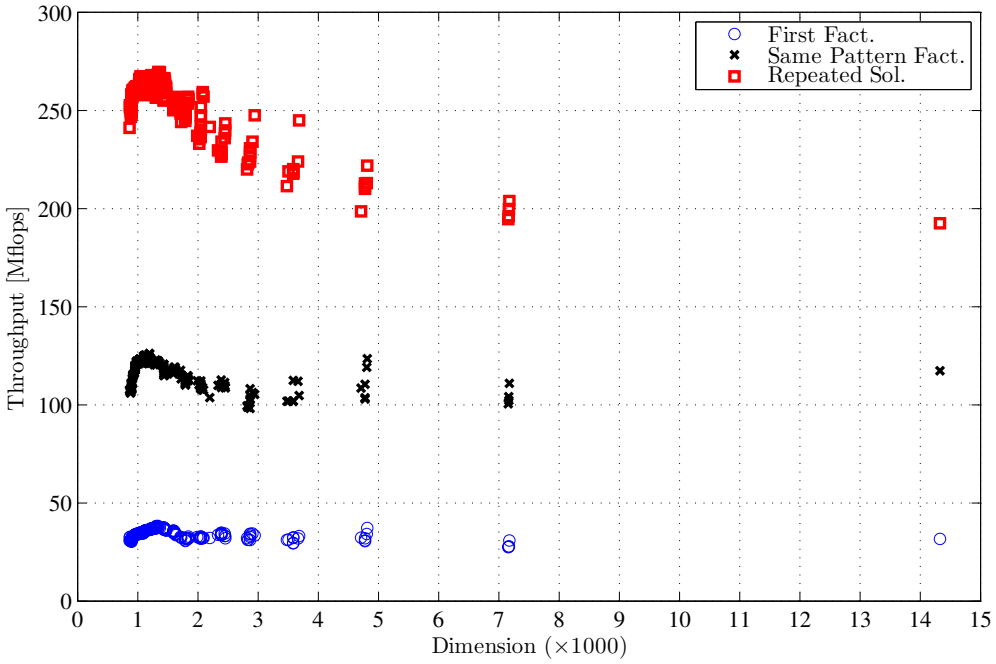


Figure 3.7. Sparse operations throughput.

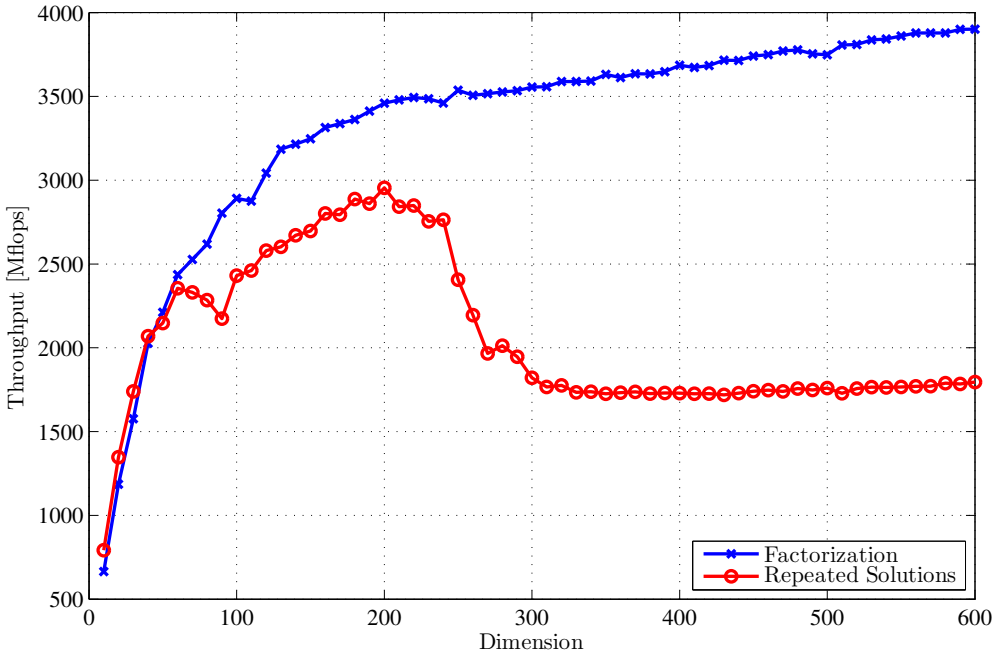


Figure 3.8. Dense operations throughput.

Table 3.2. Data fitting summary for the dense operations.

Factorization ($R^2 = 0.99975$)		
Component	Value	C.I. (95%)*
l^3	6.5066×10^{-10}	$\pm 5.2271 \times 10^{-12}$
l^2	2.7058×10^{-8}	$\pm 1.1134 \times 10^{-9}$
l	8.8535×10^{-8}	$\pm 1.9420 \times 10^{-8}$
Repeated Solution ($R^2 = 0.99855$)		
Component	Value	C.I. (95%)*
l^2	4.9757×10^{-9}	$\pm 1.9051 \times 10^{-10}$
l	-2.2719×10^{-7}	$\pm 9.1490 \times 10^{-8}$
1	2.8222×10^{-6}	$\pm 8.9642 \times 10^{-7}$

* 95% confidence interval for the parameters estimates.

3.3.3 Communication Libraries Benchmark

As previously discussed in Section 3.2.3, in order to properly characterize a network, according to the PLogP model (Kielmann et al., 2000), the latency L, sending overhead $o_s(m)$, receiving overhead $o_r(m)$ and inter-message gap $g(m)$ are required. The underlying hardware is not the only aspect to influence these parameters, but also the software layer responsible for handling the interprocess communications.

The *Message Passage Interface* (MPI) was chosen for the implementation of the communication operations required by the network-based MATE algorithm. This choice is mainly due to the fact that, since its creation in 1994, the MPI Standard (2008) have become the *de facto* standard in high-performance computing industry, which specifies the interfaces to a number of point-to-point and collective inter-processes communication kernels. As such, parallel MPI-based programs are portable and can be easily compiled against many hardware-specific MPI implementation.

Among the several MPI implementations publicly available, the well-known MPICH2 (Argonne National Laboratory, 2007) and NMPI (NICEVT, 2005) libraries were selected. The MPICH2 is a widely used MPI implementation, developed by the Argonne National Laboratory, USA, which provides the entire MPI 2.1 standard (MPI Standard, 2008) for shared memory and a number of distributed network stacks, such as TCP, InfiniBand, SCTP and Myrinet. The NMPI library, based on the previous MPICH2, implements the MPI standard over SCI networks. These two ready-to-use MPI libraries are examples of highly tested and hardware-specialized pieces of software, which abstract the programming from the hardware, significantly reducing the development time.

Since two network types are available in the employed parallel computing environment, the acquisition of the network parameters, latency L, sending overhead o_s, receiving overhead o_r and minimum inter-message gap g, not only provides the means to predict the network performance, but also enables comparing the two networks. Results of the benchmark, summarized in Appendix C, are depicted in Figures 3.9 and 3.11 for the SCI and Ethernet-based networks available in the UBC's Power System Engineering Group's cluster.

The minimum inter-message gaps g, shown in Figure 3.9 for both Ethernet and SCI networks, are inversely related to the maximum bandwith of the network. Since the parameter $g(m)$ denotes the minimum time necessary to send an m-byte message to another process, the bandwidth of the network can be calculated as the ratio between the message size m and its associated gap $g(m)$, as shown in (3.32).

$$B(m) = \frac{m}{g(m)} \tag{3.32}$$

The inter-message $g(m)$ plots, shown in Figure 3.11, suggest that, for the Ethernet and SCI networks, $g(m)$ can be approximated by a piece-wise linear function, such as (3.33). In the expression, the coefficient G represents the gap per byte in a long message, according to the LogGP model, proposed by (Alexandrov et al., 1995), whereas g_0 coincides with the estimated minimum gap between zero-length messages, $g(0)$.

$$g(m) = G\,m + g_0 \tag{3.33}$$

As a consequence, for very large messages, the bandwidth $B(m)$ reaches a steady-state, i.e., the maximum network bandwidth, which is identical to the reciprocal of G, i.e., $B_{max} = \frac{1}{G}$. According to the fitted parameters for both Ethernet and SCI networks, reported in Table 3.3, the maximum bandwidth of the Gigabit Ethernet network was 122 MB/s, while the SCI yielded 182 MB/s.

The other parameters, the latency L and the overheads o_s and o_r for sending and receiving messages, respectively, were also obtained, according to the approach summarized in Appendix C and depicted in Figures 3.9and 3.10. One similarity observed for both networks is, in fact, related to some MPICH2's design strategies (Argonne National Laboratory, 2007) regarding the handling of short and large messages. Two message protocols are usually employed by MPI implementations to differentiate short and large messages, namely, the *eager* and the *rendezvous* protocols[8]. In both MPICH2 and NMPI libraries, the default threshold for the message size is 128 kB, which is characterized by sharp discontinuities in both

[8]For more information about these protocols see Appendix C.

networks benchmarks.

For the Ethernet network, the average values of o_s and o_r vary practically linearly, as observed in Figures 3.9a and 3.10. In this case, the receiving overhead o_r is slightly less than the sending overhead o_s, for messages sizes smaller than 128 kB. For large messages, however, due to the switching between eager and rendezvous message protocols, the receiving overhead o_r increases considerably, approaching the inter-message gap g, while the sending overhead o_s presents just a slight increase. Such an increase in o_r is related to the extra required handshakes between processes, in order to guarantee available memory for the transmission and readiness of the receiver.

For the SCI network, in addition to the eager and rendezvous message protocols, an extra message protocol is provided, which takes advantage of hardware-specific features for sending very short messages of less than 12 kB. In this short message protocol, the messages are encoded, so the receiver can interpret the message without communicating with the sender. This extra work required to decode short messages is captured by the faster increase rate verified for the receiving overhead o_r. As also observed in Figure 3.9b, the sending overhead o_s is coincident with the inter-message gap g, due to the error checking, included in the implementation provided by the NMPI library, which blocks the process until the transfer is complete. For the rendezvous protocol region, similarly to the Gigabit Ethernet behavior, both overheads become practically equal because of additional handshakes required between processes.

Other aspects regarding the performance of each network can be extracted comparing the sending overhead o_s and the inter-message gap g, according to Culler et al. (1996). For the Ethernet network, for instance, $o_s < g$, for all benchmarked message sizes, which shows that the hardware represents the bottleneck for the communications, since the CPU is able to process data in a much faster pace than the NIC is able to handle them. On the other hand, the SCI network presented $o_s = g$ throughout the tested message size range, which shows that the software is the bottleneck of the communications.

Regarding the latencies, they cannot be measured directly from any process, but only inferred, because it captures the time that the message spends in the network. According to the procedure summarized in Appendix C, the latency L associated with the MPICH2 implementation over the Gigabit Ethernet network was 12.8 μs. As for the SCI, because of the blocking behavior of the embedded error checking functions, the latency cannot be extracted accurately, as for the Gigabit Ethernet network, only estimates can be obtained. In this case, half of round trip timing for zero-byte messages can yield an upper limit for the latency. The measured round trip timing for zero-byte messages was about 7 μs, which limits the latency to 3.5 μs. Such a latency clearly shows the superior processing power of

Table 3.3. Parameter summary for Ethernet and SCI networks.

Ethernet (latency $L = 12.8\ \mu$s)		
Coefficient	Value	C.I. (95%)*
$g_0\,[\mu\text{s}]$	0.0 (152.68)†	±0.0 (±2020.7)†
$G\left[\frac{\mu\text{s}}{\text{kB}}\right]$	8.4944 (8.4055)†	±0.0147 (±14.077)†
SCI (latency $L = 3.5\ \mu$s)‡		
Coefficient	Value	C.I. (95%)*
$g_0\,[\mu\text{s}]$	6.6246	±0.5435
$G\left[\frac{\mu\text{s}}{\text{kB}}\right]$	5.6415	±0.0081

* 95% confidence interval for the parameters estimates.
† Parameters around brackets are fitted for messages of size greater than 128 kB.
‡ Latency upper bound (half of the round trip time of a zero-byte message).

the SCI network.

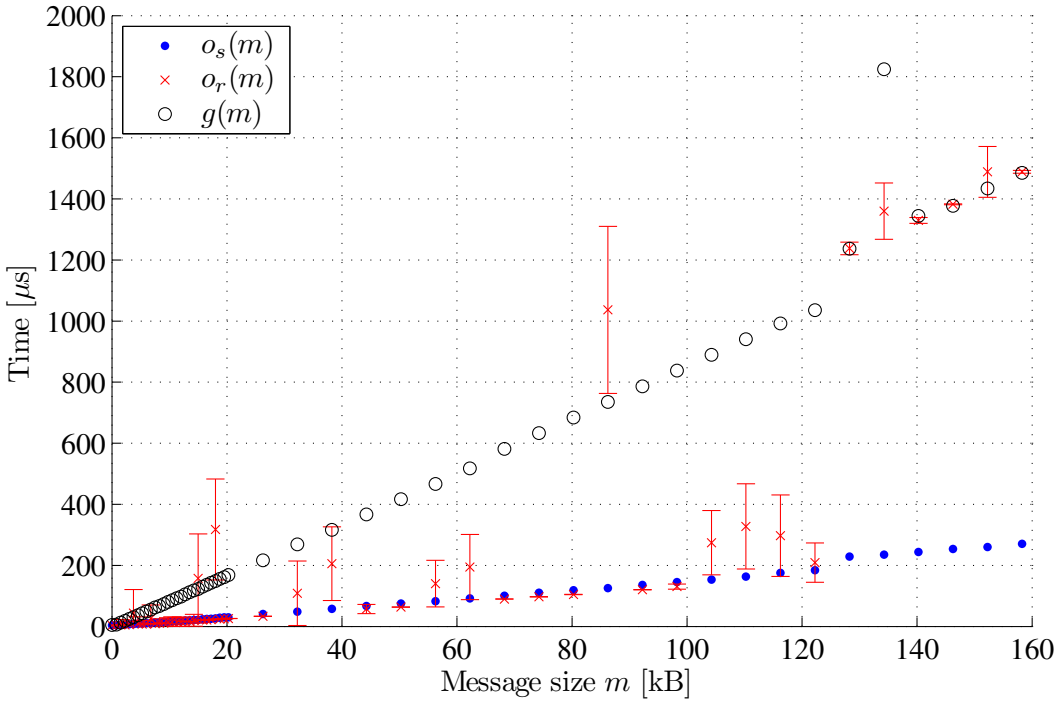
One major difference that should also be noticed is the variability of both networks, which is remarkably lower for the SCI network than for the Ethernet network, as the error bars for the receiving overheads o_r show in Figure 3.9. Such latent variability depends very much on loading conditions of network and amount connections made to a single network interface. As a consequence, in collective communications, necessary for implementing the network-base MATE algorithm, for instance, the Ethernet network is expected to present even higher variability, if messages are sent to the same process from many others simultaneously.

Network-based MATE Communication Time

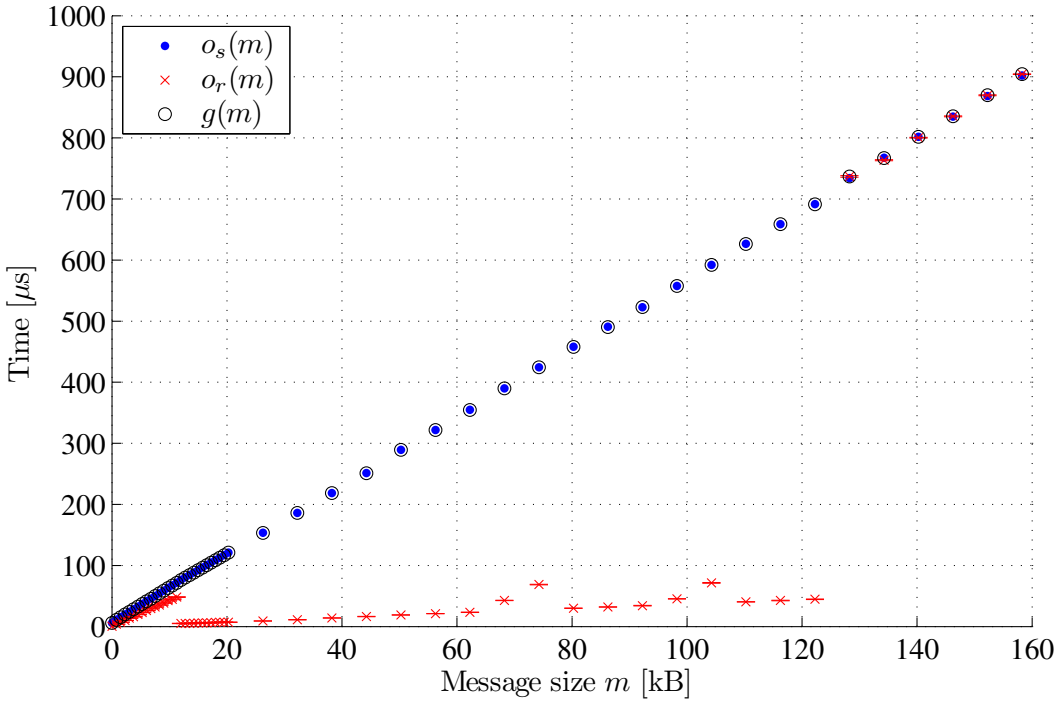
Now that all required network parameters are available, the performance of both Gigabit Ethernet and SCI networks can be compared in the context of the network-based MATE algorithm.

As previously analyzed in Section 3.2.3, the communications required by the network-based MATE algorithm involve gathering subsystems' Thévenin equivalents in the link solver and scattering link currents back to the subsystems. The timings estimates for these tasks are defined, respectively, as $T^{Thv}_{comm}(\mathbb{L})$, denoted in (3.20), and $T^{i}_{comm}(\mathbb{L})$, denoted in (3.15). Since these timings follow an identical structure, (3.34) will be used for comparing the both network interfaces.

$$T^{MATE}_{comm}(\mathbb{L}) = L + o_r(m) + o_s(m) + (p-1)\,g(m) \tag{3.34}$$



(a) Gigabit Ethernet network



(b) Scalable Coherent Interface (SCI)

Figure 3.9. Benchmark results of two MPI implementations over (a) SCI and (b) Gigabit Ethernet networks.

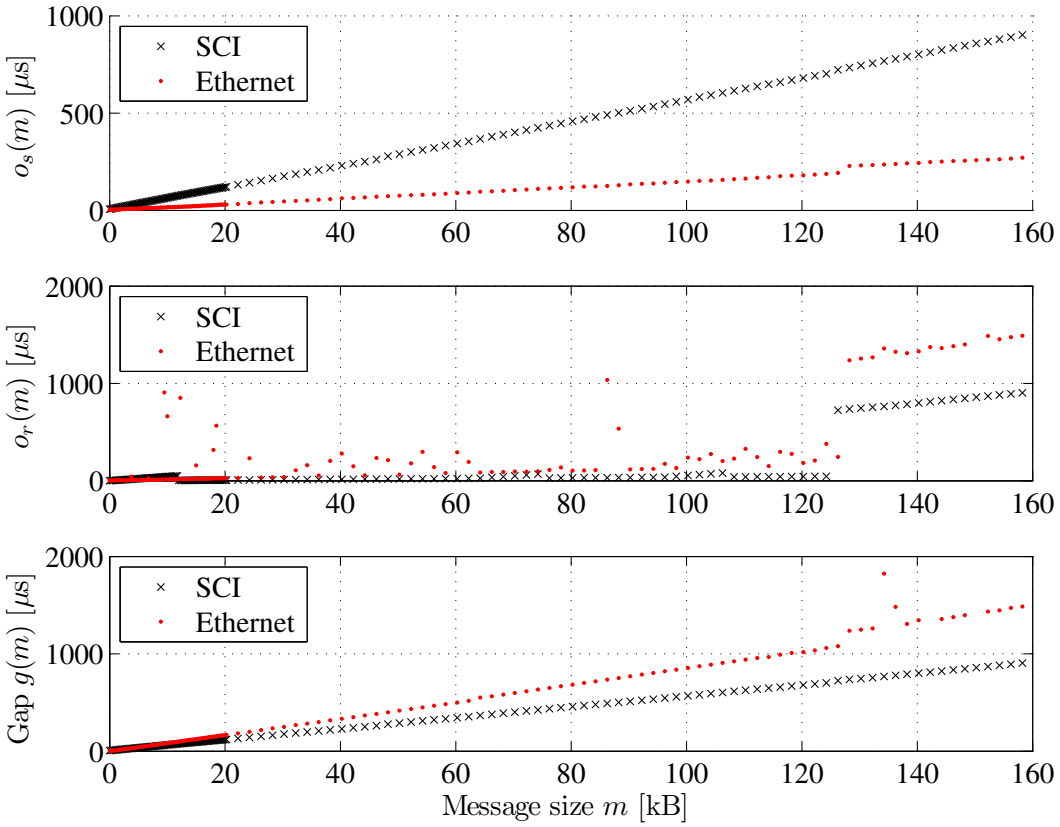


Figure 3.10. Sending and receiving overheads and inter-message gaps for MPI over SCI and Gigabit Ethernet network.

The message sizes m depend on the communication task and size of the subsystems Thévenin equivalents and number of link currents. For the Thévenin equivalents' gathering, for instance, there are two types of messages: full Thévenin equivalents, which consist of impedances and voltages, and only the Thévenin voltages. The size of the Thévenin equivalents vary on a case basis and depend mostly on the the system's topology and partitioning strategies. Although the partitioning of the system will be discussed in the next section, the subsystems' information presented in Table 3.4 show that a real power system with about 15,000 buses and partitioned into 2 up to 14 subsystems may present Thévenin equivalents of order ranging from 1 to 100. Considering now that the Thévenin equivalents are complex-valued and each complex number takes 16 B of memory, for the previous system, messages sizes ranging up to about 16 kB for the Thévenin voltages alone, and 160 kB for the full Thévenin equivalents may be expected.

In Figure 3.12, the estimated timings $T_{comm}^{MATE}(\mathbb{L})$ are plotted for 2 and 14 subsystems. In the case of two subsystems, the estimated timings for both SCI and Ethernet networks are very close to one another for messages sizes smaller that 128 kB, while the estimated

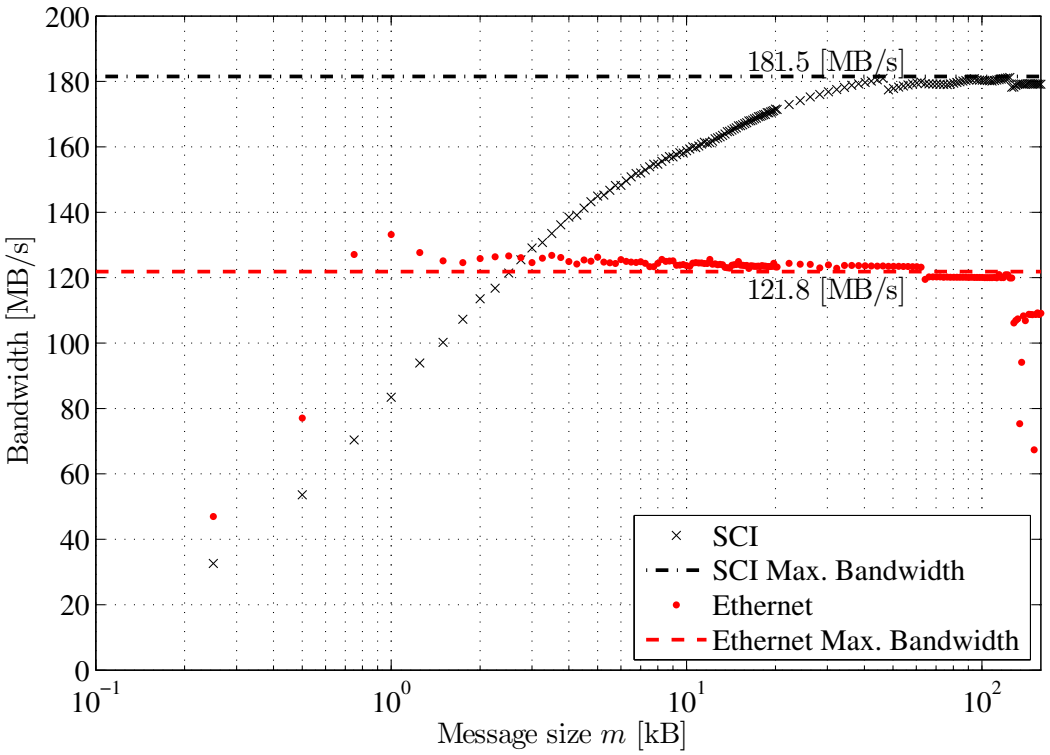


Figure 3.11. Bandwidth of MPI over SCI and Gigabit Ethernet networks.

timings for the Ethernet network exceed the ones for the SCI network in about 20%. For 14 subsystems, the SCI network is expected to present a better performance throughout the observed message size range. In this case, the estimated timings for the Gigabit Ethernet network can be up to 55% higher than the timings estimated for the SCI network.

The estimated timing $T_{comm}^{MATE}(\mathbb{L})$, defined in (3.34), however, does not consider data collision, which is true only for the SCI network (IEEE, 1993). In the case of the Ethernet networks, data collision is handled by the CSMA/CD algorithm (REFERENCE). In this algorithm, whenever the Ethernet card detects data collision, the data is retransmitted after a time delay, aka *backoff delay*, determined by the truncated binary exponential backoff algorithm[9]. In the MATE context, such a characteristic may seriously degrade the efficiency of the Thévenin equivalents gathering routine and even lock up all subsequent computations. This undesirable behavior can be explained by the many sending requests the link solver process may receive from the subsystems' processes. A condition that may even worsen this scenario is when the subsystem's computations are equally balanced and sending resquest are posted to the link solver almost simultaneously.

[9] After i collisions, a random number of slot times between 0 and $2^i - 1$ is chosen. The truncated aspect refers to the maximum number of acceptable increases before the exponentiation stops.

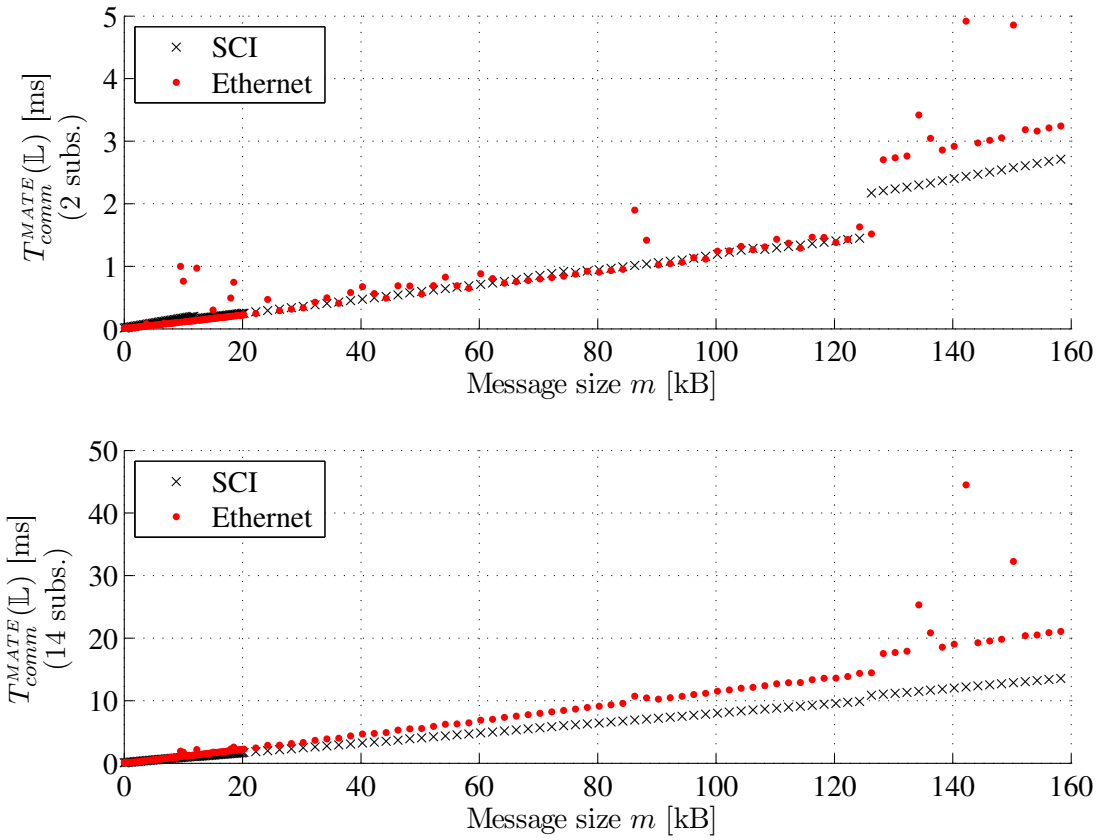


Figure 3.12. Comparison of Ethernet and SCI network's timings for the network-based MATE communications.

Due to its better performance and inherent determinism, the SCI network will employed for the network-based MATE algorithm timings presented in the next section.

3.4 Western Electricity Coordinating Council System

The Western Electricity Coordinating Council (WECC) is responsible for coordinating the bulk electric system, including generation and transmission, serving all or part of the 14 Western American States, in addition to British Columbia and Alberta in Canada. The WECC region encompasses a vast area of nearly 1.8 million square miles, which makes this system the largest and most diverse of the eight regional councils of the North American Electric Reliability Council (NERC) depicted in Figure 3.13 (`http://www.wecc.biz/wrap.php?file=wrap/about.html`).

For the present analysis, the admittance matrix which represents the WECC electric network will be employed. The WECC system presented has 14,327 buses and 16,607 branches, resulting in an admittance matrix of order 14,327 and 47,541 non-zero elements. These

numbers show that only about 0.23% of the WECC system matrix is actually filled with non-zeros, which characterizes the high level of sparsity of such large power systems. The WECC admittance matrix pattern is shown in Figure 3.14.

3.4.1 WECC System Partitioning

Load balancing and minimization of the communication volume between subsystems and number of links play a vital role when implementing the MATE algorithm. Ideally, subsystems need to be as equally sized and have as few links as possible. In such a case, the local computations (e.g., $\mathbf{Y}$ matrix factorization and $\mathbf{v}$ solution) should require about the same time to be performed for each subsystem and interface operations (e.g., multi-node Thévenin equivalents computation and communication) should be as fast and balanced as possible.

A supporting tool, METIS 4.0 (Karypis & Kumar, 1998b,a), intended for partitioning large unstructured graphs, was employed to assist in the partitioning procedure. This general purpose graph partitioner takes an undirected graph (defined by the $\mathbf{Y}$ matrix topology) as input and splits it into the requested number of partitions, so that they have nearly the same number of nodes and a minimum number of links interconnecting them.

The WECC system was partitioned from 2 to 14 subsystems using two different techniques, available in the METIS package, namely:

(a) Multilevel recursive bisection method;

(b) Multilevel k-way method.

The results for each partitioning method are presented in Table 3.4 (see page 74). Based on these results, aspects that influence the goodness of the partitions obtained with both methods will be evaluated, under the light of the MATE algorithm requirements.

Load balance among subsystems

One of the first aspects that should be discussed is how balanced the subsystems operations are. From the subsystems performance model described by (3.4) and (3.8), and later in (3.25), it can be verified that the number of buses n_k and number of border nodes b_k considerably affect the load balance among subsystems.

In this sense, for both partitioning heuristics, the subsystems average size $\mu(n_k)$ decreases according to $\frac{n}{p}$, where n is the size of the original system and p the number of subsystems, which indicates a fair level of balance among subsystems. More specifically, smaller standard deviations of n_k (given by $\sigma(n_k)$ in Table 3.4) lead to the conclusion that the subsystems

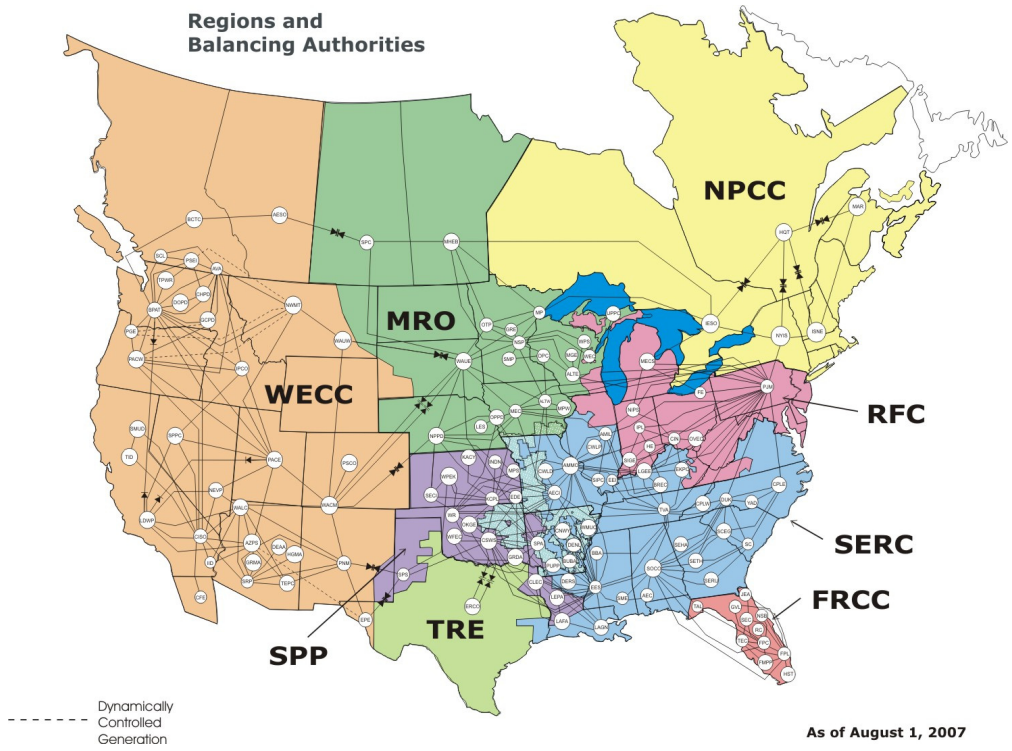


Figure 3.13. North American Electric Reliability Council (NERC) Regions: Florida Reliability Coordinating Council (FRCC), Midwest Reliability Organization (MRO), Northeast Power Coordinating Council (NPCC), Reliability First Corporation (RFC), SERC Reliability Corporation (SERC), Southwest Power Pool (SPP), Texas Regional Entity (TRE), Western Electricity Coordinating Council (WECC). (`http://www.nerc.com/page.php?cid=1%7C9%7C119`)

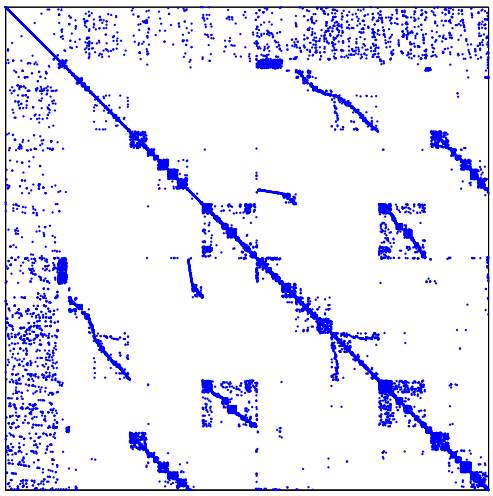


Figure 3.14. Western Electricity Coordinating Council (WECC) System admittance matrix (14,327 buses and 16,607 branches).

obtained with the recursive bisection method are generally better balanced in terms of n_k than those yielded by the k-way method.

In terms of border nodes b_k, however, the subsystems are not as balanced regardless of the partitioning method, given the bigger values of the standard deviations $\sigma(b_k)$, comparatively to the averages $\mu(b_k)$. The influence of unbalanced number of border nodes b_k on the timings of different subsystems can be explicitly shown by combining $T^{Thv}_{comp}(\mathbb{S}_k)$ and $T^{v}_{comp}(\mathbb{S}_k)$, given by (3.4) and (3.8), into the total subsystem computation time $T^{subs}_{comp}(\mathbb{S}_k)$ given below.

$$T^{subs}_{comp}(\mathbb{S}_k) = \Big[T_{1fact}(\mathbb{S}_k) + (N_z - 1)\, T_{2fact}(\mathbb{S}_k) + 2\, N_i\, T_{solv}(\mathbb{S}_k)\Big] + N_z\, b_k\, T_{solv}(\mathbb{S}_k) \tag{3.35}$$

Bearing in mind that the timings $T_{1fact}(\mathbb{S}_k)$, $T_{2fact}(\mathbb{S}_k)$ and $T_{solv}(\mathbb{S}_k)$, given by (3.5), (3.6) and (3.7), respectively, depend on well balanced quantities, such as number of nodes n_k and branch-to-node ratio ρ_k, one can induce that the term between square brackets in (3.35) will also be well balanced across all subsystems. The term dependent on the number of border nodes b_k, however, will not. Moreover, this last component is also dependent on the number of factorizations N_z the subsystems need to undergo. Thus, the less factorizations required, the less unbalanced the subsystems computations become.

Communication volume

The communication volume involved in the MATE algorithm is directly associated with the size of the multi-node Thévenin equivalents that need to be gathered in the link solver, i.e., the number of border nodes b_k. Such statement becomes evident from the timings $T^{Thv}_{comm}(\mathbb{L})$ and $T^{i}_{comm}(\mathbb{L})$ defined in (3.15) and (3.20).

Both evaluated partitioning algorithms have the number of links l interconnecting all subsystems as a minimizing function. Indirectly, minimizing such an objective function also minimizes the total number of border nodes in the system, but not necessarily the local number of border nodes. Unbalances in the size of the subsystems multi-node Thévenin equivalents are indicated by the relatively big standard deviations $\sigma(b_k)$.

For large systems, the impact of the unbalanced number of border nodes b_k will be hardly comparable to the impact of the subsystems and link solver workload. For instance, whenever systems have to be factorized often, workloads in the subsystems and link solver are $\mathcal{O}(\rho^2 n)$ and $\mathcal{O}(l^3)$, respectively, while the communication burden is $\mathcal{O}\left(c\,\overline{b}_k{}^2\right)$. Hence, as long as $\overline{b}_k{}^2 \ll \rho^2 n$ and $\overline{b}_k{}^2 \ll l^3$, computational workload will be much more significant than the communication burden.

For systems that require only a few factorizations during a simulation, the minimization of the links that straddle the subsystems can be expected to yield reasonable performance,

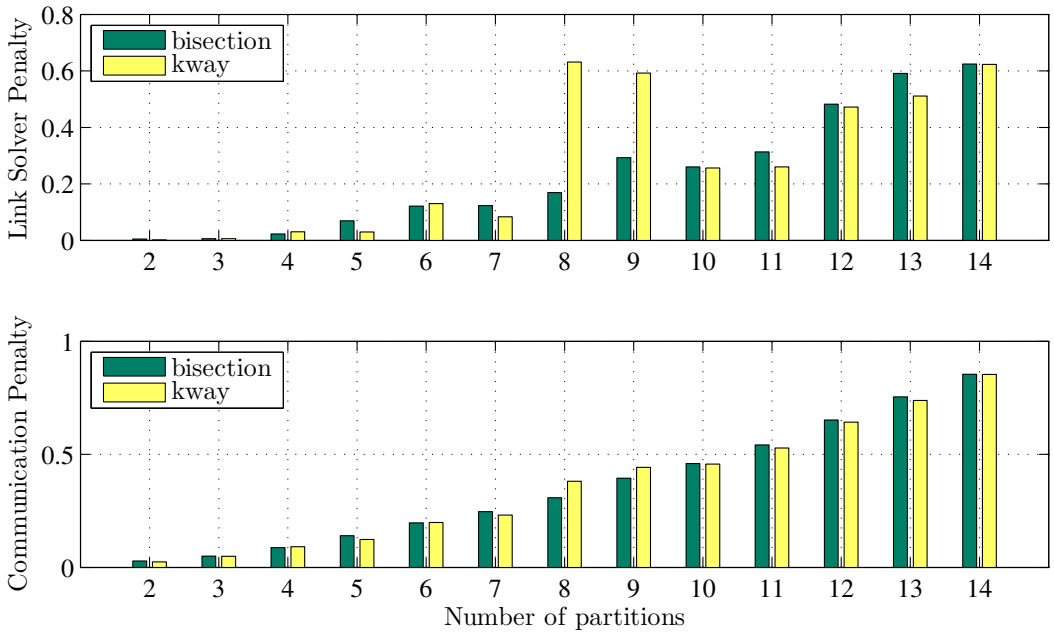


Figure 3.15. Link solver and communication penalty factors relative to the WECC system.

as long as the constraints introduced in (3.31) are satisfied. Figure 3.15 shows that the communication penalty factors remain below the unity for all adopted partitioning schemes. This fact indicates that subsystems' workload is sufficiently more significant than the communication overhead.

Sequential Link Solver

According to the computation time for the link currents $T^i_{comp}(\mathbb{L})$, defined in (3.10), the work load assigned to the link solver is minimum as long as the number of links l that straddle all subsystems is also minimum.

From Table 3.4, it can be observed that both tested partitioning algorithms are able to generate roughly similar partitions as far as the number of links l is concerned. Since none of the partitioning heuristics results in consistently smaller number of links, they can only be compared for specific number of partitions.

Additionally, given that the link solver penalty factors shown in Figure 3.15 remain below unity for all adopted partitioning schemes, one can conclude that the subsystem's workload is still sufficiently bigger than the link solver's. Therefore, for systems that need to be factorized only a few times, reasonable performance should be expected for the chosen partitioning schemes.

Table 3.4. Partitioning of the WECC system using METIS library.

(a) Multi-level recursive bisection algorithm

p	l	$\mu(n_k)$	$\sigma(n_k)$	$\underline{n_k}$	$\overline{n_k}$	$\mu(\rho_k)$	$\sigma(\rho_k)$	$\underline{\rho_k}$	$\overline{\rho_k}$	$\mu(b_k)$	$\sigma(b_k)$	$\underline{b_k}$	$\overline{b_k}$	$\mu(l_k)$	$\sigma(l_k)$	$\underline{l_k}$	$\overline{l_k}$
2	46	7163.50	0.50	7163	7164	3.74	0.05	3.69	3.78	41.00	3.00	38	44	46.00	0.00	46	46
3	43	4775.67	0.47	4775	4776	3.71	0.35	3.25	4.08	26.67	3.77	24	32	28.67	3.30	25	33
4	71	3581.75	0.43	3581	3582	3.68	0.37	3.21	4.25	32.50	7.02	25	40	35.50	8.20	27	46
5	111	2865.40	0.49	2865	2866	3.53	0.18	3.30	3.78	39.40	14.32	23	63	44.40	17.06	25	74
6	134	2387.83	1.34	2386	2390	3.51	0.24	3.27	3.91	39.83	18.13	20	62	44.67	18.28	21	66
7	125	2046.71	0.45	2046	2047	3.54	0.29	3.09	3.98	32.57	6.90	24	43	35.71	8.14	27	48
8	137	1790.88	0.33	1790	1791	3.51	0.28	3.11	3.95	29.63	11.64	11	53	34.25	13.27	12	60
9	170	1591.89	0.31	1591	1592	3.46	0.30	3.09	4.02	32.11	7.05	24	44	37.78	8.70	28	56
10	152	1432.70	0.64	1432	1434	3.49	0.30	2.99	3.93	27.00	8.26	8	42	30.40	10.08	9	50
11	159	1302.45	0.50	1302	1303	3.49	0.32	2.98	3.95	25.18	7.81	9	37	28.91	8.32	12	43
12	189	1193.92	0.28	1193	1194	3.47	0.33	3.06	3.97	27.00	8.47	6	37	31.50	10.59	7	45
13	201	1102.08	0.92	1100	1104	3.44	0.32	2.95	4.15	26.15	9.91	7	43	30.92	12.77	8	50
14	199	1023.36	0.89	1022	1025	3.43	0.40	2.85	4.05	24.21	9.01	7	41	28.43	10.93	7	52

(b) Multi-level k-way algorithm

p	l	$\mu(n_k)$	$\sigma(n_k)$	$\underline{n_k}$	$\overline{n_k}$	$\mu(\rho_k)$	$\sigma(\rho_k)$	$\underline{\rho_k}$	$\overline{\rho_k}$	$\mu(b_k)$	$\sigma(b_k)$	$\underline{b_k}$	$\overline{b_k}$	$\mu(l_k)$	$\sigma(l_k)$	$\underline{l_k}$	$\overline{l_k}$
2	25	7163.50	9.50	7154	7173	3.63	0.07	3.56	3.69	23.00	0.00	23	23	25.00	0.00	25	25
3	43	4775.67	47.25	4709	4813	3.67	0.06	3.58	3.73	25.00	9.42	13	36	28.67	11.26	13	39
4	82	3581.75	91.46	3478	3682	3.62	0.12	3.45	3.77	37.25	10.47	28	55	41.00	11.25	31	60
5	73	2865.40	49.83	2816	2943	3.70	0.20	3.49	4.05	25.40	15.21	5	49	29.20	17.45	5	56
6	139	2387.83	96.07	2195	2455	3.54	0.13	3.37	3.73	41.00	21.35	17	78	46.33	26.00	18	92
7	103	2046.71	35.62	1987	2092	3.55	0.19	3.30	3.94	25.86	10.51	5	40	29.43	12.00	6	47
8	265	1790.88	59.00	1710	1840	3.45	0.19	3.19	3.84	55.38	31.00	20	111	66.25	38.37	23	133
9	242	1591.89	79.90	1368	1631	3.47	0.14	3.24	3.63	45.89	18.50	10	68	53.78	21.09	10	78
10	151	1432.70	74.37	1212	1474	3.57	0.25	3.14	4.08	26.50	14.54	4	48	30.20	15.20	7	51
11	145	1302.45	36.25	1242	1344	3.56	0.25	3.13	3.89	22.73	6.84	11	34	26.36	7.67	14	38
12	187	1193.92	59.19	1014	1229	3.52	0.18	3.15	3.74	25.67	15.11	3	61	31.17	19.93	3	80
13	187	1102.08	18.66	1073	1132	3.47	0.26	3.08	3.91	24.08	11.87	3	48	28.77	15.67	3	58
14	199	1023.36	27.70	954	1049	3.47	0.21	3.16	4.02	24.29	9.84	4	40	28.43	11.95	5	48

- $\overline{x_k}$ means the maximum value of x_k;
- $\underline{x_k}$ means the minimum value of x_k;
- $\mu(x_k)$ means the average value of x_k;
- $\sigma(x_k)$ means the standard deviation of x_k.

3.4.2 Timings and Performance Predictions for the WECC System

To predict timings and performance, 1000 solutions of the WECC system were considered, i.e., $N_i = 1000$, and three distinct number of factorizations $N_z = \{1, 100, 500\}$.

The predicted and measured timings are shown in Figures 3.17, 3.18 and 3.19. As the number of partitions increases, so does the number of links interconnecting the subsystems, which, in turn, become smaller (see Table 3.4). Such behavior helps reducing the participation of the subsystems computations in the total time, while boosting up the workload assigned to the link solver. Due to the significantly increased workload in the link solver, the timing ceases to decrease for higher number of partitions. However, since on modern processors, dense matrix computations are more efficient than sparse computations in terms of Mflops, the computational overhead incurred by the link solver increases at a much slower pace than the subsystems' workload decreases. This fact helps the present MATE implementation achieve up to 6 times speedup with respect to the sequential SuperLU algorithm, as shown in Figure 3.23.

Figures 3.17, 3.18 and 3.19 also show that as the number of factorizations N_z increase, the link solver operations increase with $\mathcal{O}\left(N_z\, l^3\right)$, therefore, in a much faster rate than the subsystems sparse operations which increase with a function $\mathcal{O}\left(\frac{n}{p} N_z\right)$, where n represents the number of buses in the original untorn WECC system. This explains the lower speedups achieved when the system needs to be factorized many times during the simulations, regardless of the number of partitions.

As mentioned before in Section 3.4.1, the communication between subsystems and links solver has little impact on the global timings, regardless of the number of factorizations, although more data is exchanged whenever a factorization in the subsystems level is required. This is because of the massive amount of computations required in comparison with the amount of data needed to be exchanged among processes.

In Figures 3.20, 3.21 and 3.22, the MATE algorithm performance is illustrated for the case when the WECC system is torn apart into 14 subsystems by means of the multilevel recursive bisection partitioning method. In these graphs, the processes participating in the solution are represented in the x-axis, where process 0 is related to the link solver and the rest to the subsystems. Observe that for fewer factorizations, the partitions obtained with the METIS library show good computational balance, which deteriorates as the number of required factorizations grows. This tendency is explained by the fact that the number of border nodes b_k are not well balanced across the subsystems (see Table 3.4 for details), which makes the multi-node Thévenin equivalent in distinct subsystems very different.

As for the performance metrics, speedup and efficiency were measured with respect to the SuperLU sparse solver timings. Results are depicted in Figure 3.23. It can be observed

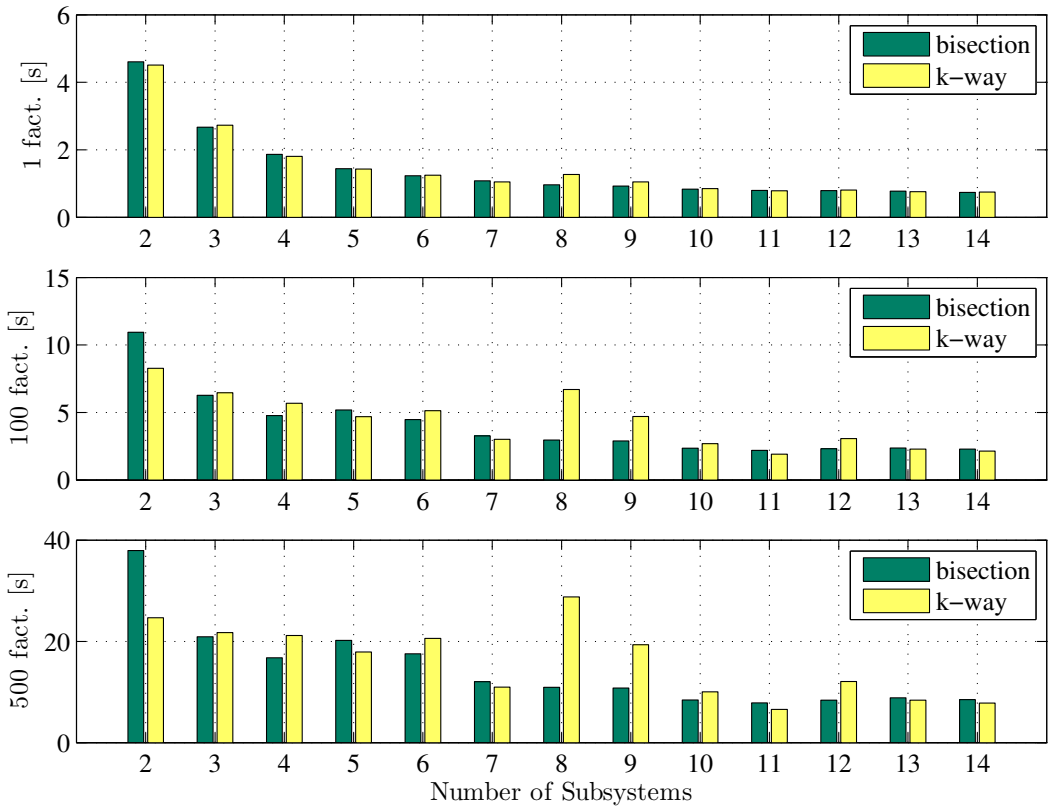
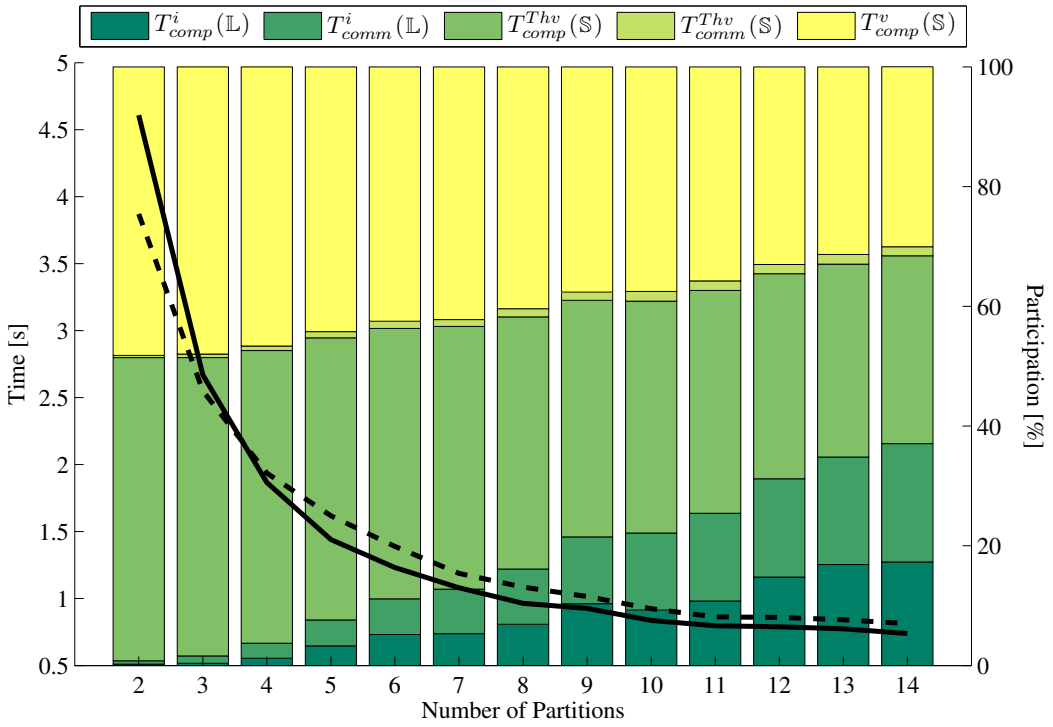


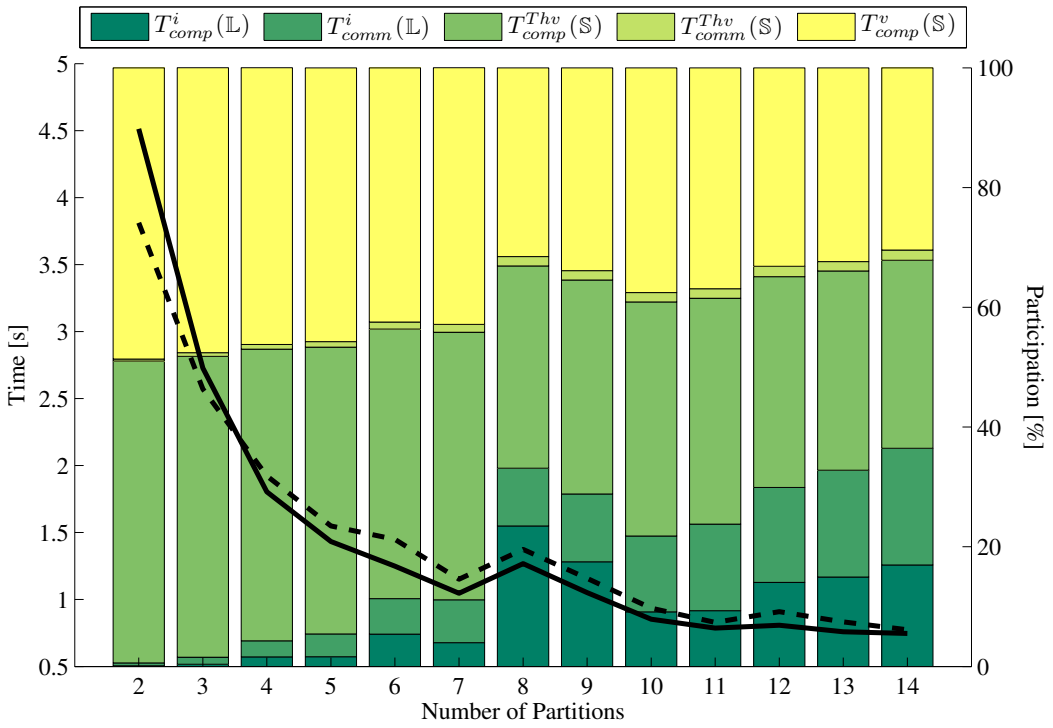
Figure 3.16. Comparison between MATE timings for multilevel recursive bisection and multilevel k-way partitioning algorithms.

that, in case when the WECC system required just a few factorizations (less than 5% of the steps), the network-based MATE algorithm achieved about 6 times speedup with respect to the sequential SuperLU solver, when 14 partitions were considered. For many factorizations (more than 10% of the steps), however, the algorithm speedup degrades considerably to approximately 3 times for $N_z = 100$, and 2 times for $N_z = 500$. Although maximum speedup happened for the 14-subsystems case, the maximum efficiency was observed for a lower number of partitions. For small number of factorizations, a slightly higher than 50% efficiency was observed, while lower values were registered for the cases with many factorizations, i.e., less that 30% for $N_z = 100$ and less that 20% for $N_z = 500$.

Lastly, timings obtained with both partitioning algorithms are compared in Figure 3.16. Both multilevel recursive bisection and k-way deliver similar performance to the MATE algorithm when only a few factorizations are required. The differences between the employed partitioning heuristics become evident for higher number of factorizations. These differences in performance come as a result of larger number of links and local border nodes, as it can observed in Table 3.4, when the WECC system is partitioned into 2, 8 and 9 subsystems.

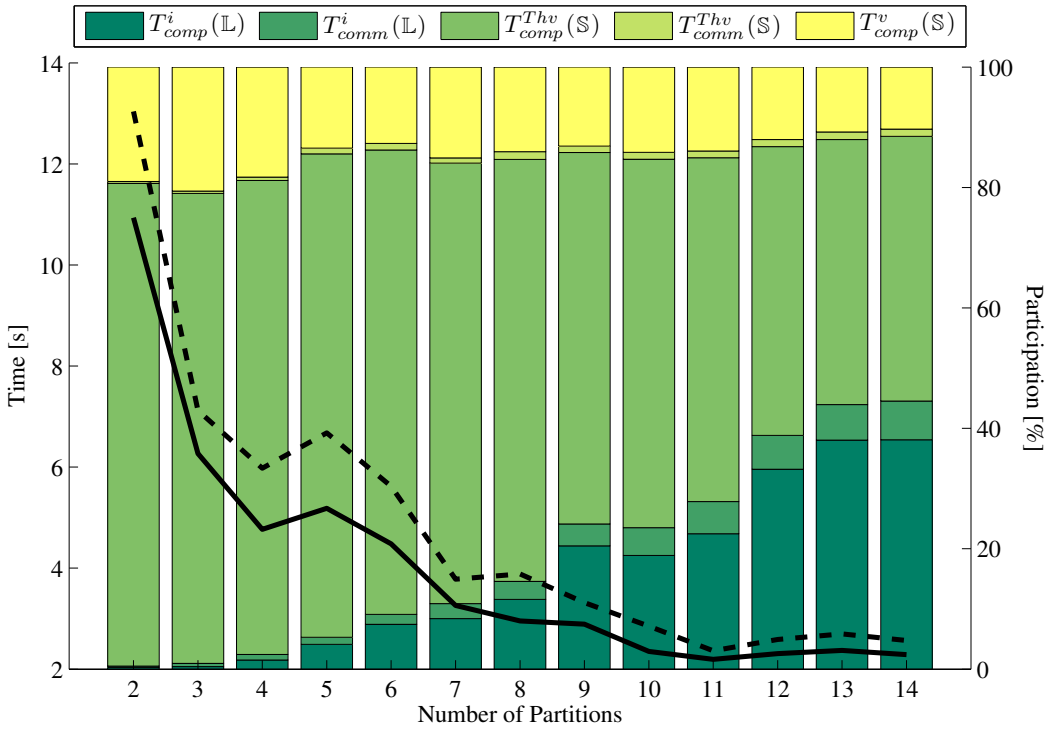


(a) multilevel recursive bisection

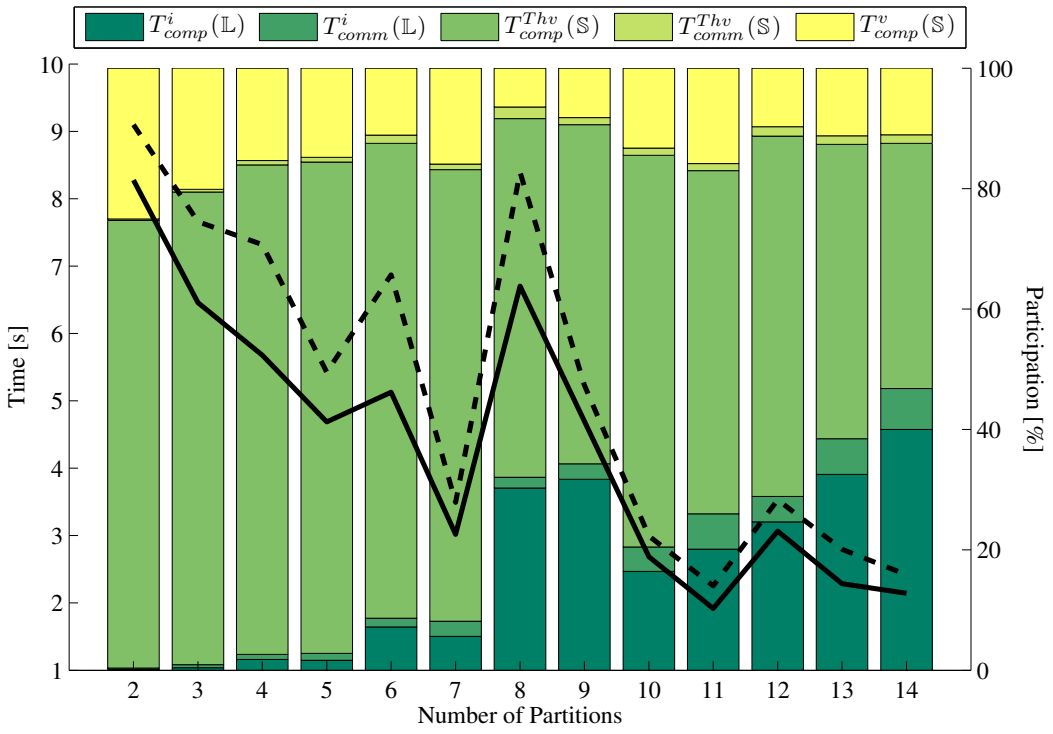


(b) multilevel k-way partitioning

Figure 3.17. MATE predicted and measured timings for the solution of the WECC system for 1000 steps, 1 factorization and different partitioning strategies.

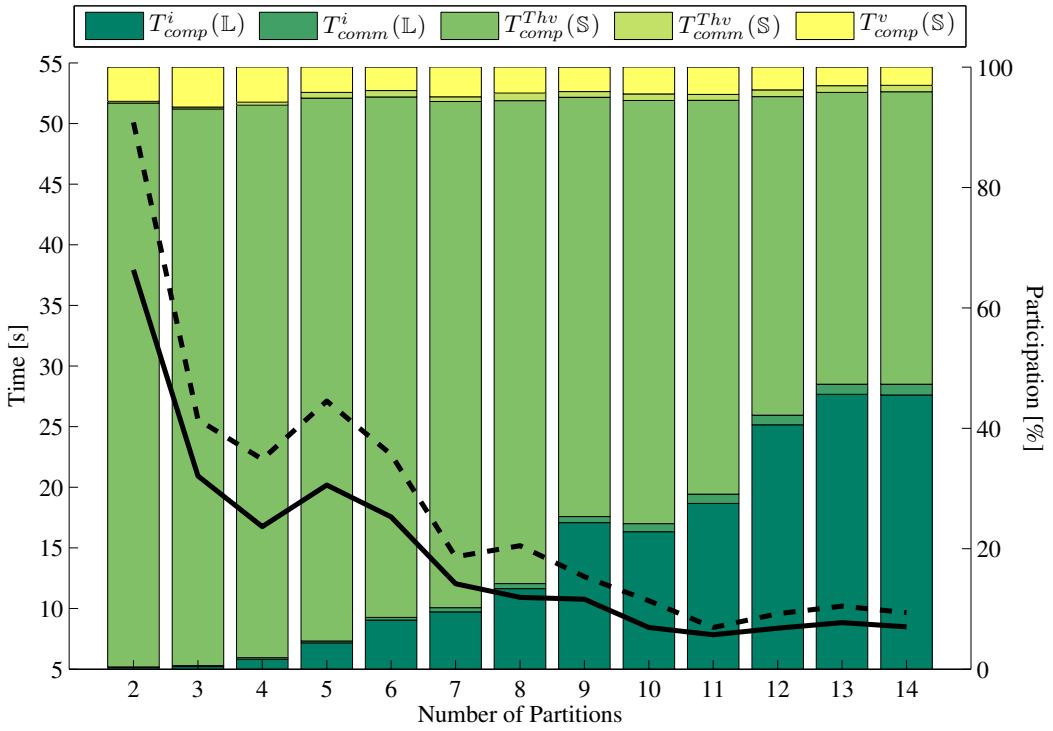


(a) multilevel recursive bisection

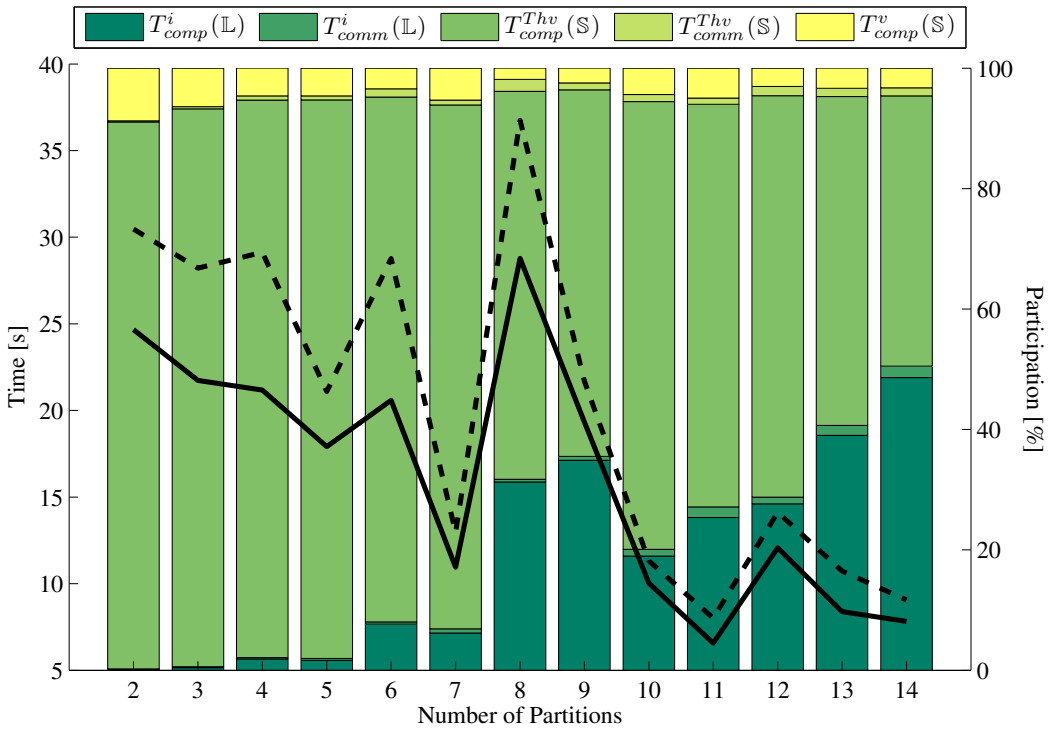


(b) multilevel k-way partitioning

Figure 3.18. MATE predicted and measured timings for the solution of the WECC system for 1000 steps, 100 factorizations and different partitioning strategies.

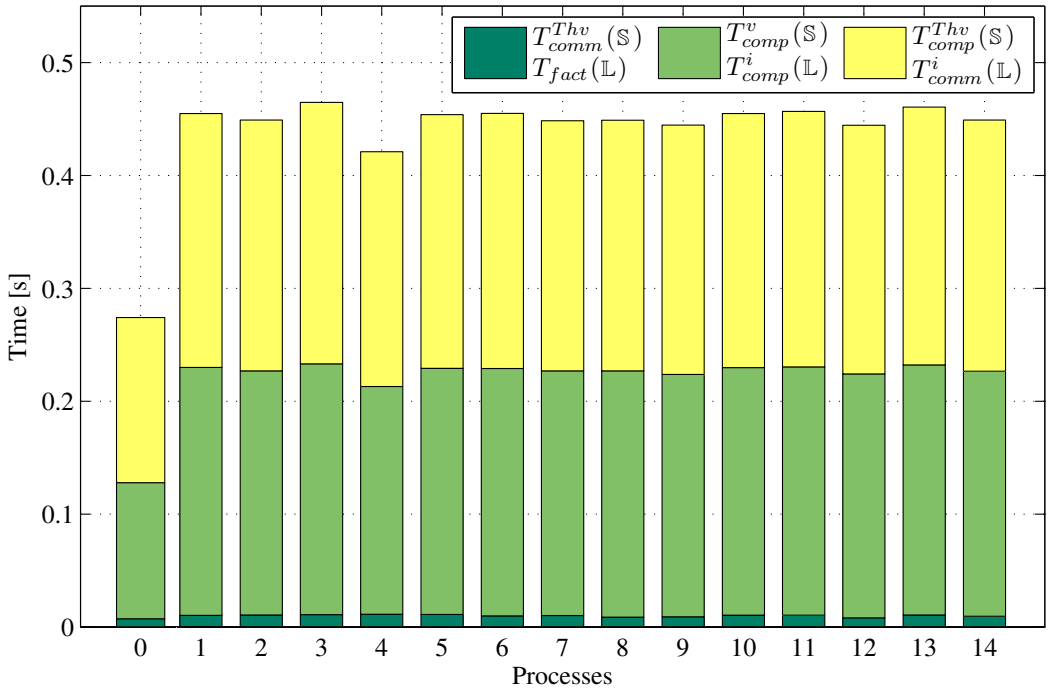


(a) multilevel recursive bisection

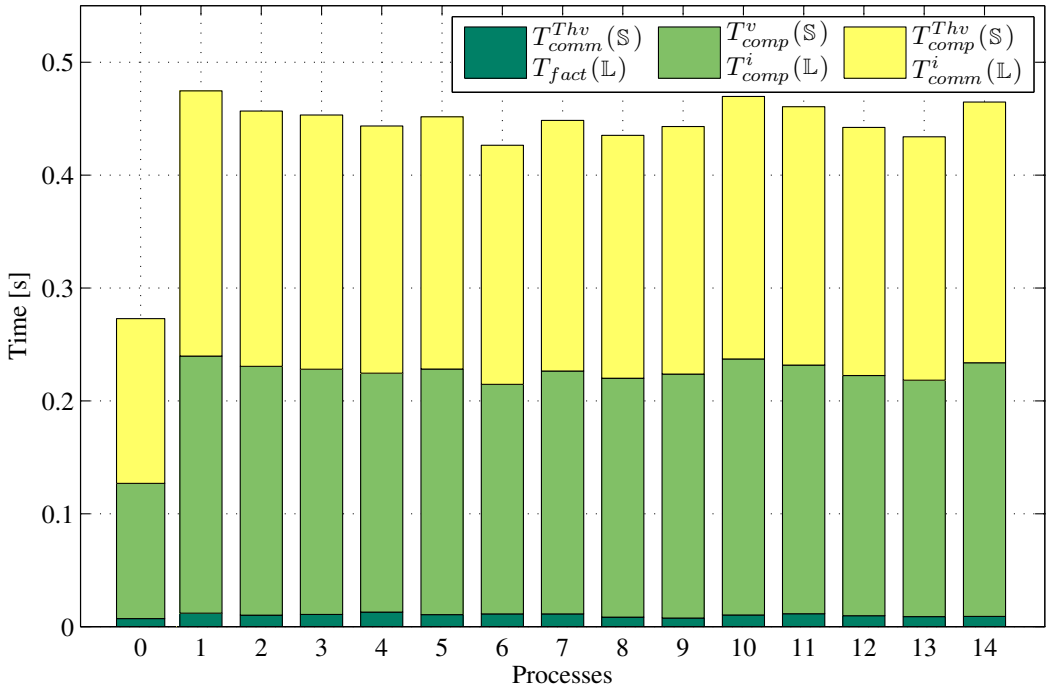


(b) multilevel k-way partitioning

Figure 3.19. MATE predicted and measured timings for the solution of the WECC system for 1000 steps, 500 factorizations and different partitioning strategies.

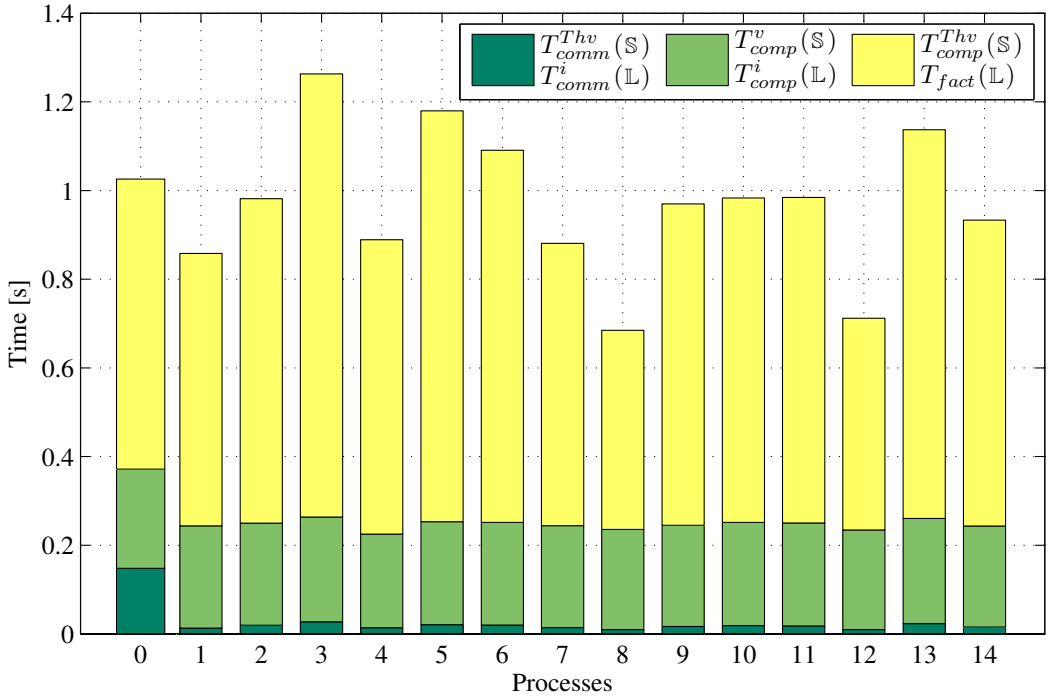


(a) multilevel recursive bisection

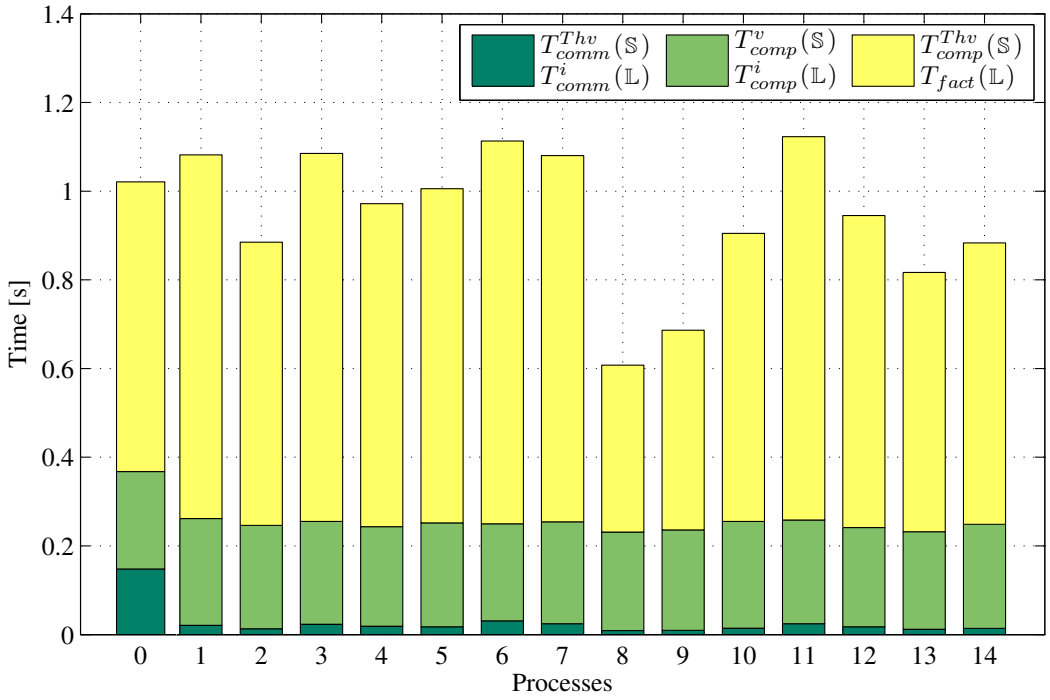


(b) multilevel k-way partitioning

Figure 3.20. MATE timings for the solution of the WECC system partitioned in 14 subsystems for 1000 steps, 1 factorization and different partitioning strategies. (Process 0 relates to $\mathbb{L}$ and processes 1-14 with $\mathbb{S}$)

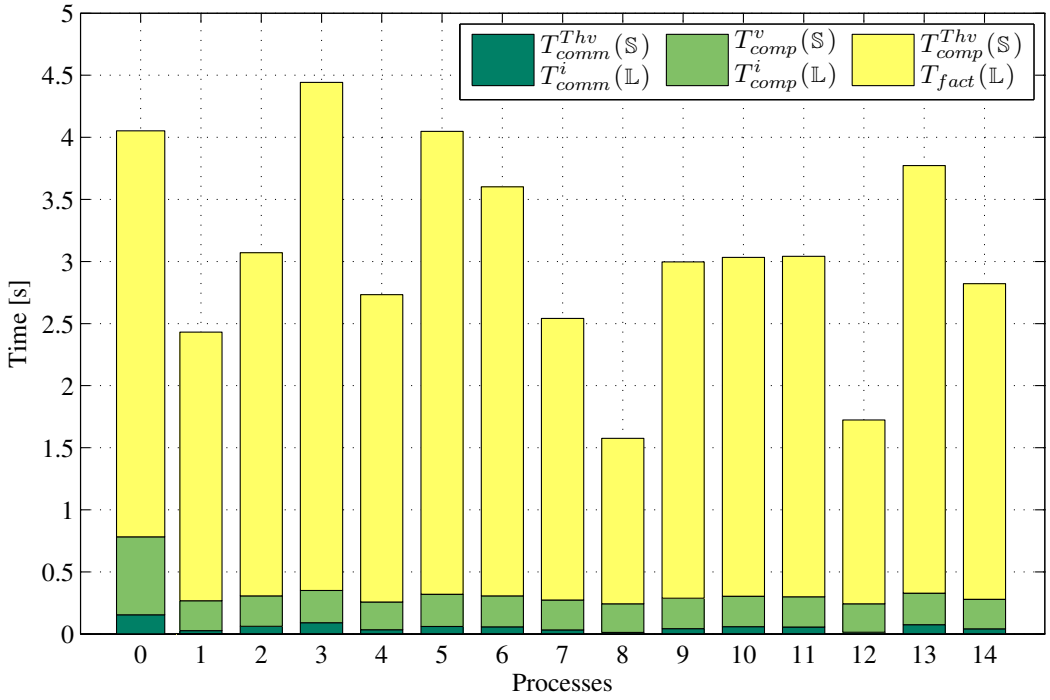


(a) multilevel recursive bisection

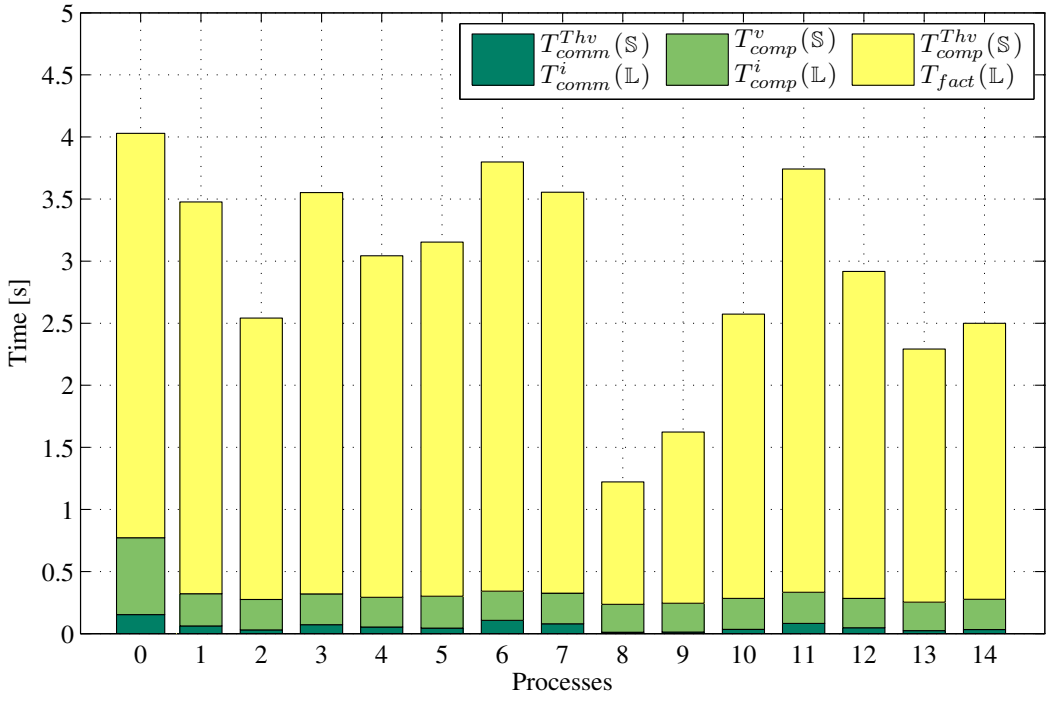


(b) multilevel k-way partitioning

Figure 3.21. MATE timings for the solution of the WECC system partitioned in 14 subsystems for 1000 steps, 100 factorizations and different partitioning strategies. (Process 0 relates to $\mathbb{L}$ and processes 1-14 with $\mathbb{S}$)

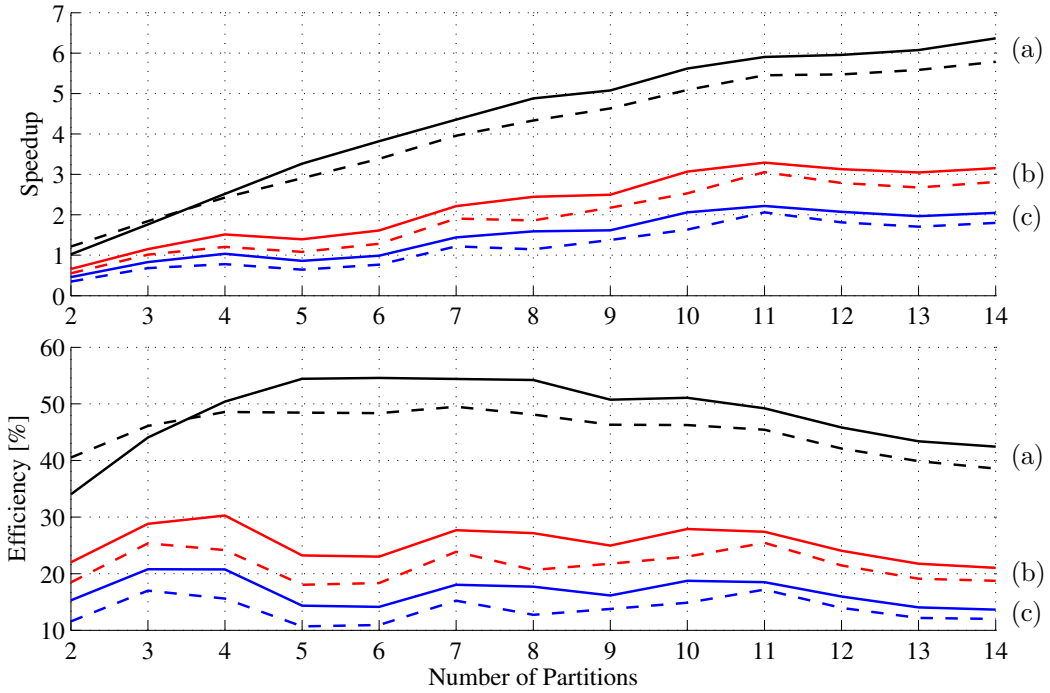


(a) multilevel recursive bisection

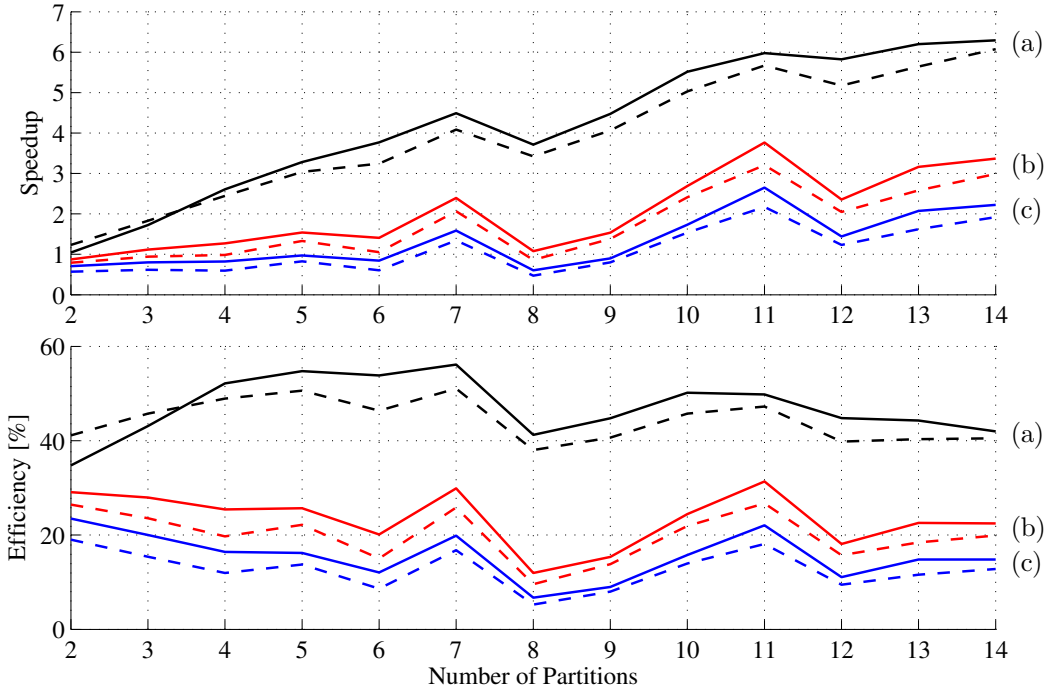


(b)

Figure 3.22. MATE timings for the solution of the WECC system partitioned in 14 subsystems for 1000 steps, 500 factorizations and different partitioning strategies. (Process 0 relates to $\mathbb{L}$ and processes 1-14 with $\mathbb{S}$)



(a) Multilevel recursive bisection partitioning



(b) Multilevel k-way partitioning

Figure 3.23. MATE performance metrics for the solution of the WECC system for 1000 steps and (a) 1, (b) 100 and (c) 500 factorizations.

3.5 Conclusion

In this chapter, a performance model for a specific implementation of the network-based MATE algorithm is discussed, along with an application on a real large power system, the Western Electricity Coordinating Council (WECC) system, with about 15,000 buses.

The approach adopted on the present implementation was to combine traditional sparsity techniques with dense operations, in order to obtain an optimized parallel MATE-based linear solver.

From the developed performance model, it was shown that for systems, which need to be repeatedly solved, but factorized only a few times, the theoretical speedup of the network-based MATE algorithm with respect to traditional sparsity-oriented linear solvers is $\frac{p}{2}$, for a system partitioned into p subsystems. As a consequence, for the same type of systems, the efficiency cannot exceed 50%. Nonetheless, compared with speedup and efficiency reported in the literature (see Section 1.2), these values can be considered competitive with other parallel algorithms employed in solving sparse linear solvers, even on massive supercomputers.

The concept of link solver and communication penalty factors, introduced in (3.31), provides a normalized measure of the influence of the link solver computations as well as the communication overhead in the network-based MATE algorithm, with respect to the subsystems workload. In the future, such factors could be even used as objective functions in the system partitioning phase.

For the WECC system, the speedup was shown to approximate fairly well the theoretical one, with obtained speedups slightly higher than 6 times when solving the WECC system partitioned into 14 subsystems (see Figure 3.23). Concerning the efficiency of the algorithm, it was kept very close to the theoretical 50% for all tested partitioning schemes using the multi-level recursive bisection and multi-level k-way methods (Karypis & Kumar, 1998b).

Chapter 4

MATE-based Parallel Transient Stability

According to Kundur (1994), transient stability is the ability of the power system to maintain synchronism when subjected to a severe transient disturbance such as a fault on transmission facilities, loss of generation or loss of load. As such, the main objective of transient stability simulations is to detect critical operating conditions under which power systems would loose synchronism due to a large disturbance.

In engineering applications, it is frequently desirable to make many system response simulations to calculate, for example, the effects of different fault locations and types, automatic switching, initial power system operating states, different network, machine and control-system characteristics (Stott, 1979). Such studies provide vital information for system design, operation planning and, more recently, real-time assessment of large power systems. The sheer volume of computation required by such studies, however, imposes severe constraints on the size and complexity of the systems that can be analyzed by means of on-line Transient Stability Assessment (TSA) tools. Hence, speeding up transient stability calculations is one way of improving the reliability of operation and security of modern power systems (Andersson et al., 2005).

In order to address the need of faster TSA tools, a parallel transient stability simulator, which employs the network-based MATE algorithm as the back-end for network solutions, will be presented. This implementation differs from other parallel transient stability programs in the sense that it is inherently designed for distributed-memory computing systems. This is because the network-based MATE algorithm's features allow the complete separation of subsystems. This approach, in addition to avoiding extraneous data movement between central databases and subsystems processes, also allows the system data to be partitioned according to the topology of the network under study. This characteristic also improves the parallelism of other tasks, required by the transient stability simulator, such as dynamic models integration.

For the sake of the comparisons, one acclaimed transient stability solution technique implementation is discussed in its sequential and parallel versions. Timing results of the actual implementations, along with speedup and efficiency of the solution, are shown for a reduced version of the Brazilian National Interconnected System, with 1916 buses, 2788 branches and 79 generators.

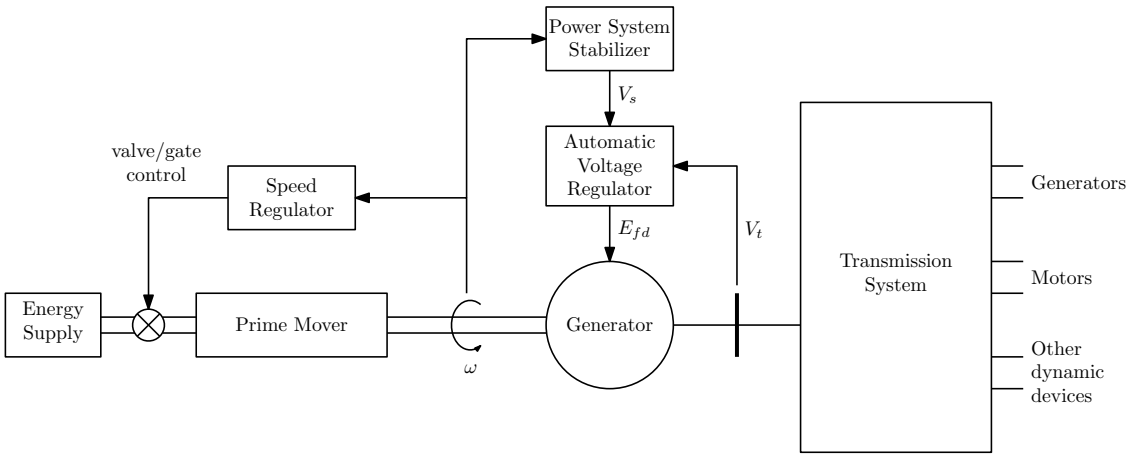


Figure 4.1. Basic structure of a power system model for transient stability analysis.

4.1 Transient Stability Problem

According to the literature (Dommel & Sato, 1972; Stott, 1979; Kundur, 1994), the power system transient stability problem is modelled by means of a set of non-linear differential-algebraic equations (DAEs), which can be summarized as follows:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{v}) \tag{4.1a}$$

$$\mathbf{Y}\,\mathbf{v} = \mathbf{i}(\mathbf{x}, \mathbf{v}) \tag{4.1b}$$

where, $\mathbf{x}$ represents a vector with dynamic variables (or, state variables), whose first derivatives $\dot{\mathbf{x}}$ are defined by a vector function $\mathbf{f}$, normally dependent on $\mathbf{x}$ themselves and the vector with nodal voltages $\mathbf{v}$. In addition, $\mathbf{i}$ represents a vector function that defines the nodal current injections, which also depend on the variable states $\mathbf{x}$ and the nodal voltages $\mathbf{v}$. Lastly, $\mathbf{Y}$ represents the complex-valued nodal admittance matrix of the system under study. For illustration purposes, a basic power system structure for transient stability studies is also given in Figure 4.1.

In this formulation, the set of dynamic equations (4.1a) comprises all non-linear differential equations related to generators and their associated prime movers (e.g., hydro and steam turbines) and controllers (e.g, automatic voltage regulators, speed governors and power system stabilizers), motors, FACTS devices (e.g, HVDC systems and SVCs), as well as any other dynamic device (e.g., dynamic loads). The structure of the dynamic equations strongly depends on the models used for each of the aforementioned components, which, ultimately, define the strength of the non-linearities included in the system as well as its overall response time characteristics.

The set of static equations (4.1b), in turn, includes all algebraic complex-valued nodal equations associated with the passive transmission network. This network is mainly composed by transformers and transmission lines, stator of generators and motors, and voltage-dependent loads. Depending on the models used for the loads, the algebraic equations may also become non-linear. The interface between the dynamic and static equations in (4.1) is accomplished by the voltage-dependent algebraic equations associated with each device connected to the passive network. The stator equations of generators figure as the most important interface equations, as far as the transient stability is concerned.

4.1.1 Transient Stability Solution Techniques

Many solution variations for the problem defined by (4.1) have been described in the literature. These variations combine different integration methods for the dynamic equations (4.1a), different solution methods for algebraic equations (4.1b), and different manners of interfacing these dynamic and algebraic equations. A number of these methods are summarized in (Stott, 1979).

All proposed variations, however, fall into two major categories, which are closely related to the interface between the dynamic and algebraic equations. These categories are the (a) *alternating* (or *partitioned*) and (b) *simultaneous* solution approaches.

As pointed out by Stott (1979), the alternating approach is the most traditional method and adopted by many industrial-grade transient stability programs. In addition, since the alternating method relies on repeated network solutions during a simulation, it presents itself as an ideal option for direct application of the network-based MATE algorithm, analyzed in the previous chapters. Therefore, the alternating algorithm will detailed in the sequence, while more information with respect to simultaneous solution approach can be found in Appendix B.

Alternating Solution Approach

The alternating solution method is characterized by the fact that the dynamic equations (4.1a) are integrated separately from the algebraic equations (4.1b) solution. In addition, the integration method considered will affect the way the interface between both set of equations is realized.

In case of an explicit integration method, such as explicit Runge-Kutta methods, the derivatives $\dot{\mathbf{x}}$ calculated at previous time steps are used for computing new values of $\mathbf{x}$. Subsequently, $\mathbf{x}$ is used for the the network computation, which turns out to be iterative in case non-impedance loads are used. A nice feature provided by explicit integration formulas

lies on the fact that both sets of dynamic and algebraic equations are truly isolated from one another and, their interface solution is non-iterative. On the other hand, explicit integration methods are weakly stable and often require very small time steps.

In case of an implicit integration rule, such as Trapezoidal integration rule, extrapolation techniques are used for predicting node voltages $\mathbf{v}$, which are then employed during the integration procedure, which yields an approximate $\mathbf{x}$. With the approximate values of $\mathbf{x}$ and $\mathbf{v}$, the current injections $\mathbf{i}$ can be calculated and used for computing an updated set of voltages $\mathbf{v}$, which are later employed in the updating of the dynamic equations (4.1a). In comparison with the explicit integration methods, their implicit counterparts present improved numerical robustness and convergence characteristics.

In order to properly choose an implicit integration method for the solution of the transient stability problem, requirements of power systems need to be firstly considered. Power systems related problems usually accept numerical errors in the order of a few percent, which diminishes the importance of the order of the integration method. The implicit method, however, has to present strong numerical stability and, therefore, not accumulate errors during a simulation. Moreover, the implicit integration method should never produce stable solutions to unstable problems. Bearing this requirements in mind, the Trapezoidal integration rule has been proved to be reliable, reasonably accurate and computationally inexpensive, in comparison to higher order implicit formulas (Dommel & Sato, 1972; Stott, 1979).

As such, the set of ordinary differential equations (4.1a) is first discretized according to the Trapezoidal integration rule. This procedure is shown in (4.2).

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(t-\Delta t) + \int_{t-\Delta t}^{t} \mathbf{f}\big(\mathbf{x}(\xi), \mathbf{v}(\xi)\big) d\xi \approx \\ &\approx \mathbf{x}(t-\Delta t) + \frac{\Delta t}{2}\Big[\mathbf{f}\big(\mathbf{x}(t), \mathbf{v}(t)\big) + \mathbf{f}\big(\mathbf{x}(t-\Delta t), \mathbf{v}(t-\Delta t)\big)\Big]\end{aligned} \tag{4.2}$$

Collecting the present and past values yields (4.3), where $\mathbf{x}_h(t)$ represents the *history term* of the state vector $\mathbf{x}$, which is known at time t.

$$\mathbf{x}(t) = \frac{\Delta t}{2}\,\mathbf{f}\big(\mathbf{x}(t), \mathbf{v}(t)\big) + \mathbf{x}_h(t) \tag{4.3a}$$

$$\mathbf{x}_h(t) = \mathbf{x}(t-\Delta t) + \frac{\Delta t}{2}\,\mathbf{f}\big(\mathbf{x}(t-\Delta t), \mathbf{v}(t-\Delta t)\big) \tag{4.3b}$$

Combining the original set of algebraic equations defined in (4.1b) with the one produced by the integration procedure, given in (4.3), results the alternating solution method given in

(4.4).

$$\mathbf{x}^m = \frac{\Delta t}{2}\,\mathbf{f}\left(\mathbf{x}^m, \mathbf{v}^m\right) + \mathbf{x}_h \tag{4.4a}$$

$$\mathbf{Y}\,\mathbf{v}^{m+1} = \mathbf{i}\left(\mathbf{x}^m, \mathbf{v}^m\right) \tag{4.4b}$$

where $\mathbf{x}_h(t)$, defined in (4.3b), is computed from previous values of $\mathbf{x}$ and $\mathbf{v}$, and $m = 0, 1, 2, \ldots$ stands for the iteration count.

The equations described in (4.4) describe the iterative process needed for the solution at a given time t. The process starts with the computation of $\mathbf{v}^m$ by extrapolation on a few immediate past values of $\mathbf{v}$. Once $\mathbf{v}^m$ is computed, a new iterate $\mathbf{x}^m$ is obtained from (4.4a). Then, applying the previous $\mathbf{x}^m$ and $\mathbf{v}^m$ to calculate $\mathbf{i}\left(\mathbf{x}^k, \mathbf{v}^k\right)$, (4.4b) can be solved for $\mathbf{v}^{m+1}$, which can be used to start a new iteration. The iterative process continues until the difference between $\mathbf{v}^{m+1}$ and $\mathbf{v}^m$ is negligible.

4.1.2 Transient Stability Models

The set of equations (4.1), which define the transient stability problem, is highly dependent on the power system under investigation as well as its associated models, as discussed in Section 4.1.

However, before delving into modelling of power system devices for transient stability studies, one need to keep in mind that the choice of the models has to be in agreement with the time scale of interest. As described by Kundur (1994), transient stability can be subdivided into three categories: *short-term* (up to 10 seconds), *mid-term* (from 10 seconds to a few minutes) and *long-term stability* (from a few minutes to tens of minutes). The modelling for different time scales may differ considerably. For instance, in short-term stability studies, generators and their respective automatic controls (voltage and speed) in addition to the transmission system with the loads are often enough, while for long-term stability, boiler dynamics of thermal power plants, penstock and conduit dynamics of hydro plants, automatic generation control, frequency deviation effects on loads and network, excitation limiters, among others, have to be included.

Moreover, bearing in mind that the present implementation aims at showing the feasibility of parallel transient stability computations employing the network-based MATE (see Chapters 2 and 3), the modelling requirements for the short-term stability will considered. Thus, the most basic models for short-term transient stability, the generators and the transmission system, will be introduced.

Transmission Network

The basic consideration in transient stability studies, as far as the transmission system modelling goes, is that the frequency of the whole system is assumed to be near constant and close to the rated system frequency, such as 50 and 60 Hz. Moreover, the very fast decaying network transients are seldom of concern in the time scale of interest. Therefore, network dynamics are usually neglected in transient stability studies. Under such considerations, the basic transmission system is modelled by a large set of nodal algebraic equations which relate voltages at all buses of the system and their current injections. This set of equations is commonly represented by a large sparse and usually complex-valued admittance matrix $\mathbf{Y}$, which is topologically symmetric but numerically unsymmetric. Furthermore, in transient stability problems, the $\mathbf{Y}$ matrix is also constant between topological changes, such as short-circuits and line tripping.

The basic elements considered in the transmission network model are the transmission lines, the transformers and the buses.

(a) *Transmission lines*

A lumped π circuit, depicted in Figure 4.2, is normally employed for characterization of the transmission lines in transient stability studies. According to (Kundur, 1994), the parameters of the transmission line π circuit are given as follows:

$$Z_s = Z_C \sinh(\gamma l) \tag{4.5a}$$

$$Y_{sh} = \frac{2}{Z_C} \tanh\left(\frac{\gamma l}{2}\right) \tag{4.5b}$$

and

$$Z_C = \sqrt{\frac{R + j\omega_s L}{G + j\omega_s C}} \tag{4.6}$$

$$\gamma = \sqrt{(R + j\omega_s L)(G + j\omega_s C)} \tag{4.7}$$

where

Z_C = Characteristic impedance, in [Ω]

γ = Propagation constant, in [m^{-1}]

l = length of the line, in [m]

R = series resistance, in [$\frac{\Omega}{\mathrm{m}}$]

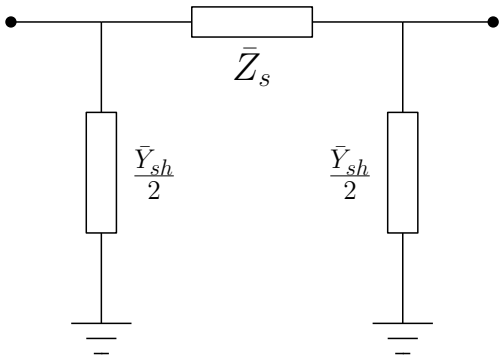


Figure 4.2. Equivalent π circuit of a transmission line

L = series inductance, in $[\frac{\mathrm{H}}{\mathrm{m}}]$

G = shunt conductance, in $[\frac{\mho}{\mathrm{m}}]$

C = shunt capacitance, in $[\frac{\mathrm{F}}{\mathrm{m}}]$

(b) *Transformers*

The single line diagram of a 2-winding transformer is shown in Figure 4.3a. In this representation the parameter a is the per unit turns ratio and $\bar{Y}_T = \frac{1}{\bar{Z}_T}$ the nominal per unit series admittance of transformer.

As presented in (Kundur, 1994), the associated π circuit of the present 2-winding transformer is depicted in Figure 4.3b and defined by the paremeters given in (4.8).

$$\bar{Y}_t = \frac{\bar{Y}_T}{a} \qquad \text{[p.u.]} \qquad (4.8a)$$

$$\bar{Y}_p = \left(\frac{1-a}{a^2}\right)\bar{Y}_T \qquad \text{[p.u.]} \qquad (4.8b)$$

$$\bar{Y}_s = \left(\frac{a-1}{a}\right)\bar{Y}_T \qquad \text{[p.u.]} \qquad (4.8c)$$

(c) *Load buses*

Modelling a load bus is a very complicated task due to the variety and sazonality of the loads that can be connected to a same bus, such as lamps, refrigerators, heaters, compressors, motors, computers and so forth. And, even if all load models were known, implementing all the millions of loads often supplied by power systems would render the transient stability simulations infeasible. Therefore, it is common practice adopting composite models for the load buses as seen from the power system.

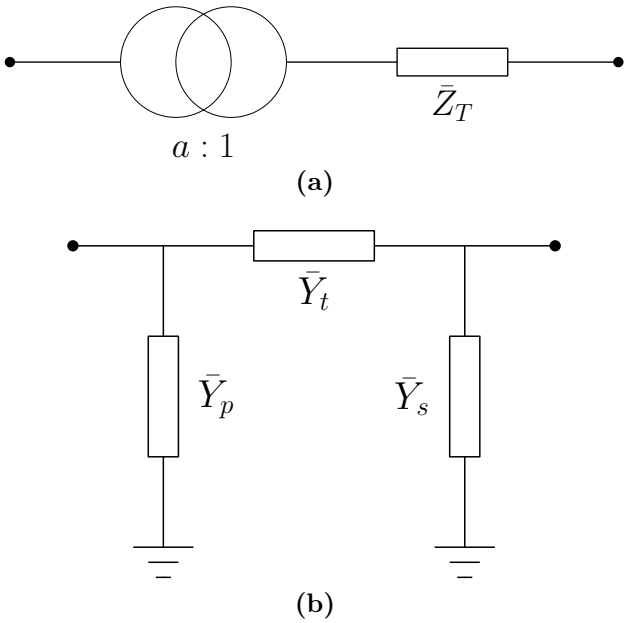


Figure 4.3. Transformer with off-nominal ratio and its π circuit representation.

One type of model that is widely used in power system industry is the polynomial model, as described in (4.9). The coefficients p's and q's represent the proportion of the components of constant power (p_1 and q_1), constant current (p_2 and q_2) and constant impedance (p_3 and q_3).

$$P_L = P_0 \left(p_1 + p_2\, V + p_3\, V^2\right) \qquad \text{[p.u.]} \tag{4.9a}$$

$$Q_L = Q_0 \left(q_1 + q_2\, V + q_3\, V^2\right) \qquad \text{[p.u.]} \tag{4.9b}$$

where $p_1 + p_2 + p_3 = q_1 + q_2 + q_3 = 1$.

In order to interface the foregoing composite load model with the network in the alternating method, as summarized by (4.4b), each component of the load needs to be interfaced with the network through an impedance or current injection or both.

In the case of the constant current component, its associated injection $\bar{I}_L^i$ is calculated from the steady-state condition, as shown in (4.10), and included in the right-hand side vector, **i**, of the system of equations defined in (4.4b).

$$\bar{I}_L^i = \frac{p_1\, P_0 - j\, q_1\, Q_0}{\bar{V}_0^*} \qquad \text{[p.u.]} \tag{4.10}$$

For the constant impedance component, similarly to the constant current case, the load admittance $\bar{Y}_L^z$ is calculated from the steady-state solution of the network by means of the expression denoted in (4.11). This constant admittance is then included in the admittance

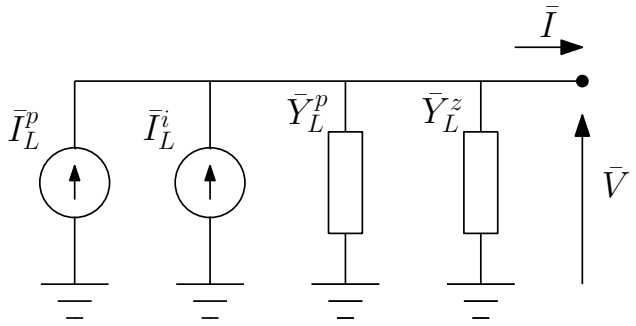


Figure 4.4. Equivalent circuit of the polynomial load model

matrix that describe the transmission system.

$$\bar{Y}_L^z = \frac{p_2\, P_0 - j\, q_2\, Q_0}{{V_0}^2} \qquad \text{[p.u.]} \tag{4.11}$$

For the constant power components, a Norton equivalent is used (Arrillaga & Watson, 2001), where the admittance $\bar{Y}_L^p$ is connected in parallel with a current injection $\bar{I}_L^p$. In this case, the current $\bar{I}_L^p$ provides a power adjustment for changes around the power initially drawn by the admittance $\bar{Y}_L^p$, so the specified power is enforced. Similarly to the constant impedance case, $\bar{Y}_L^p$ is also included in the admittance matrix of the system.

$$\bar{I}_L^p = \left(\bar{Y}_L^p - \frac{\bar{S}^*}{V^2}\right)\bar{V} \qquad \text{[p.u.]} \tag{4.12a}$$

$$\bar{Y}_L^p = (p_3\, P_0 - j\, q_3\, Q_0) \qquad \text{[p.u.]} \tag{4.12b}$$

Synchronous Generator

Synchronous machines are commonly developed in the machine rotor *qd0* reference frame (Kundur, 1994; Krause et al., 2002; Anderson et al., 2003). The full synchronous machine model describe both stator and rotor dynamics, in addition to the mechanical dynamics. However, since the very fast decaying network transients are seldom of concern in the time scale of interest, both network and the stator dynamics are usually neglected in transient stability studies. Under such consideration, the stator equations become algebraic, while the rotor equations are kept dynamic (Kundur, 1994).

Another important assumption made when modelling synchronous machines for transient stability studies is that the electrical frequency ω_e of the machines only slightly deviates from the rated frequency of the system ω_s. Under such conditions, the electromechanical torque

T_e and power P_e of the machine, when given in p.u. on the machine base quantities, are interchangeable, i.e., $T_e \approx P_e$.

There is a vast literature on synchronous generators models applied to the transient stability problem (Kundur & Dandeno, 1983; Kundur, 1994; Anderson et al., 2003), hence, only a summary of the classical and transient models will be presented. The extension of the transient to the subtransient model is straightforward and is also covered in the referenced literature.

(a) *Mechanical Equations*

The mechanical equations, given in (4.13), describe the electrical frequency ω_e of the machine and angle between the net magnetic flux in machine gap and the magnetic flux induced in the field winding, also known as the load angle of the machine δ.

$$\dot{\delta} = \omega_s\, \omega \qquad \left[\tfrac{\text{rad}}{\text{s}}\right] \qquad (4.13a)$$

$$\dot{\omega} = \frac{1}{2H}\left[P_m - P_e - D\omega\right] \qquad \left[\tfrac{\text{p.u.}}{s}\right] \qquad (4.13b)$$

$$\omega = \frac{\omega_e - \omega_s}{\omega_s} \qquad [\text{p.u.}] \qquad (4.13c)$$

where

δ = load angle, in [rad]

ω = frequency deviation from the rated electrical frequency ω_s, in [p.u.]

ω_e = electrical speed or frequency, in $\left[\tfrac{\text{rad}}{\text{s}}\right]$

ω_s = system rated frequency, in $\left[\tfrac{\text{rad}}{\text{s}}\right]$

P_m = mechanical power, in [p.u.]

P_e = electrical power, in [p.u.]

H = inertia constant, in [s]

D = damping factor, in [p.u.]

Applying the Trapezoidal integration rule to (4.13) yields the new set of algebraic equations given in (4.14).

$$\delta(t) = k_{\delta\omega}\,\omega(t) + \delta_h(t) \qquad \text{[rad]} \tag{4.14a}$$

$$\omega(t) = k_{\omega P}\big[P_m(t) - P_e(t)\big] + \omega_h(t) \qquad \text{[p.u.]} \tag{4.14b}$$

$$\delta_h(t) = \delta(t-\Delta t) + k_{\delta\omega}\,\omega(t-\Delta t) \qquad \text{[rad]} \tag{4.14c}$$

$$\omega_h(t) = k_{\omega P}\big[P_m(t-\Delta t) - P_e(t-\Delta t)\big] + k_{\omega h}\,\omega(t-\Delta t) \qquad \text{[p.u.]} \tag{4.14d}$$

where

$$k_{\delta\omega} = \frac{\omega_s \Delta t}{2} \qquad k_{\omega P} = \frac{\Delta t}{4H + D\,\Delta t} \qquad k_{\omega h} = \frac{4H - D\,\Delta t}{4H + D\,\Delta t}$$

(b) *Electrical Equations*

The algebraic transient equations associated with stator are given in (4.15), while the dynamic equations which govern the transient voltages induced onto the stator are presented in (4.16). These equations are defined in the *qd* reference frame of the machine, depicted in Figure 4.5.

$$e'_q - v_q = r_a i_q - x'_d i_d \qquad \text{[p.u.]} \tag{4.15a}$$

$$e'_d - v_d = r_a i_d + x'_q i_q \qquad \text{[p.u.]} \tag{4.15b}$$

$$\frac{de'_q}{dt} = \frac{1}{T'_{do}}\big[e_{fd} + (x_d - x'_d)\,i_d - e'_q\big] \qquad \left[\tfrac{\text{p.u.}}{\text{s}}\right] \tag{4.16a}$$

$$\frac{de'_d}{dt} = \frac{1}{T'_{qo}}\big[-\left(x_q - x'_q\right) i_q - e'_d\big] \qquad \left[\tfrac{\text{p.u.}}{\text{s}}\right] \tag{4.16b}$$

and

$$P_e = e'_d\,i_d + e'_q\,i_q + \left(x'_d - x'_q\right)\,i_q\,i_d \qquad \text{[p.u.]} \tag{4.17}$$

where

v_q, v_d = q and d-axis voltages at the terminal of the machine, in [p.u.]

i_q, i_d = q and d-axis stator currents, in [p.u.]

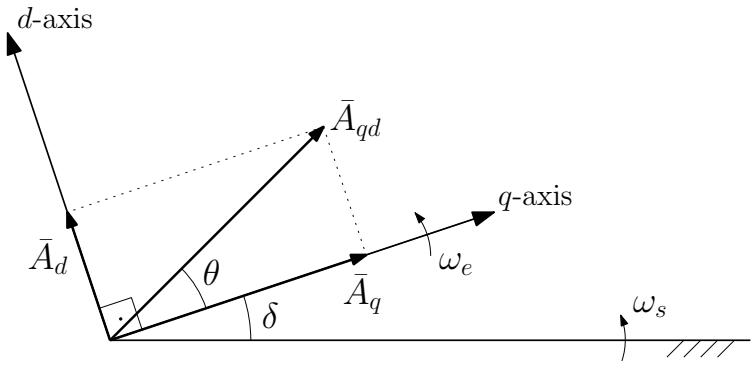


Figure 4.5. Synchronous machine qd reference frame and system reference frame

e'_q, e'_d = q and d-axis transient voltages induced at the stator of the machine, in [p.u.]

e_{fd} = excitation field voltage referenced to the stator of the machine, in [p.u.]

P_e = electrical power transferred to the machine shaft, in [p.u.]

x'_q, x'_d = q and d-axis transient reactances of the machine, in [p.u.]

T'_{qo}, T'_{do} = q and d-axis open-circuit time constants of the machine, in [p.u.]

r_a = stator resistance, in [p.u.]

Again, applying the Trapezoidal integration rule to (4.16) results the set of algebraic equations given in (4.18).

$$e'_q(t) = k_{q1}\, e_{fd}(t) + k_{q2}\, i_d(t) + e'_{qh}(t) \quad \text{[p.u.]} \tag{4.18a}$$

$$e'_d(t) = k_{d2}\, i_q(t) + e'_{dh}(t) \quad \text{[p.u.]} \tag{4.18b}$$

$$e'_{qh}(t) = k_{q1}\, e_{fd}(t-\Delta t) + k_{q2}\, i_d(t-\Delta t) + k_{q3}\, e'_q(t-\Delta t) \quad \text{[p.u.]} \tag{4.18c}$$

$$e'_{dh}(t) = k_{d2}\, i_d(t-\Delta t) + k_{d3}\, e'_d(t-\Delta t) \quad \text{[p.u.]} \tag{4.18d}$$

where

$$k_{q1} = \frac{\Delta t}{(2T'_{do} + \Delta t)} \qquad k_{q2} = k_{q1}\,(x_d - x'_d) \qquad k_{q3} = \frac{2T'_{do} - \Delta t}{2T'_{do} + \Delta t}$$

$$k_{d1} = \frac{\Delta t}{(2T'_{qo} + \Delta t)} \qquad k_{d2} = -k_{d1}\left(x_q - x'_q\right) \qquad k_{d3} = \frac{2T'_{qo} - \Delta t}{2T'_{qo} + \Delta t}$$

In case the period of analysis is small compared with T'_{do} and the effect of the amortisseurs neglected, the machine model can be further simplified by assuming the voltage e'_q constant and ignoring the equations associated with e'_d. These assumptions eliminates all differential

equations related to the electrical characteristics of the machine (Kundur, 1994). These are the foundation of the classical synchronous machine model with constant flux linkages.

(c) *Network Interface*

From (4.4), it can be noticed that dynamic models are interfaced with the network by means of the current injections $\mathbf{i}$, which are dependent on their state variables $\mathbf{x}$ and node voltages $\mathbf{v}$. In the case of the synchronous machine, the first step in computing the current injections is solving (4.15) for i_q and i_d, which results (4.19).

$$\begin{bmatrix} i_q \\ i_d \end{bmatrix} = \frac{1}{{r_a}^2 + x'_d\, x'_q} \begin{bmatrix} r_a & x'_d \\ -x'_q & r_a \end{bmatrix} \begin{bmatrix} e'_q - v_q \\ e'_d - v_d \end{bmatrix} \tag{4.19}$$

At this point, the stator current phasor $\bar{I}$ is required for interfacing the synchronous machine with the transmission network. And, since each machine has its own qd reference frame, the currents i_q and i_s need to be converted to the synchronous reference frame of the system. For this task, consider the generic phasor $\bar{A}_{qd}$ illustrated in Figure 4.5, which is originally given in the qd reference frame. In such a case, the phasor $\bar{A}_{qd}$ can be converted to the system reference frame by means of (4.20b), which results the phasor $\bar{A}$. The procedure denoted by (4.20b) represents a rotation of $\bar{A}_{qd}$ by δ radians.

$$\bar{A}_{qd} = \bar{A}_q + \bar{A}_d \begin{cases} \bar{A}_q = a_q \\ \bar{A}_d = j\, a_d \end{cases} \tag{4.20a}$$

$$\bar{A} = \bar{A}_{qd}\, e^{j\delta} \tag{4.20b}$$

Therefore, using (4.20) and (4.19), the stator current $\bar{I}$ in the system reference frame can be found, as given in (4.21).

$$\bar{I} = \left[\left(\frac{r_a - j\, x'_q}{{r_a}^2 + x'_d\, x'_q} \right) \left(e'_q - v_q \right) - j \left(\frac{r_a - j\, x'_d}{{r_a}^2 + x'_d\, x'_q} \right) \left(e'_d - v_d \right) \right] e^{j\delta} \tag{4.21}$$

Although, (4.21) gives the complex-valued synchronous machine current injection required by (4.4b), Dommel & Sato (1972) reports that injecting the forgoing current straight into the transmission network often makes the alternating solution method non-convergent. In order to circumvent such an undesirable behavior, the authors proposed describing $\bar{I}$ in terms of a voltage source behind a fictitious admittance $\bar{Y}_M$, given in (4.22).

$$\bar{Y}_M = \frac{r_a - j\, \frac{1}{2} \left(x'_d + x'_q \right)}{{r_a}^2 + x'_d\, x'_q} \tag{4.22}$$

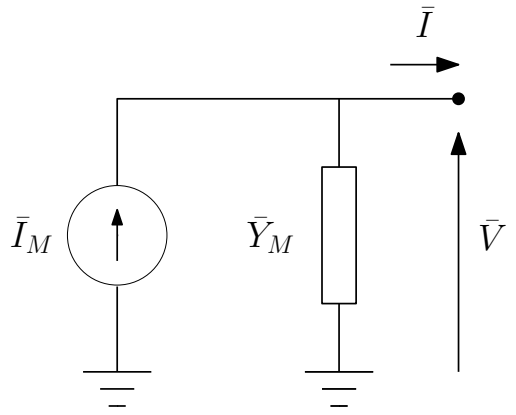


Figure 4.6. Synchronous generator equivalent circuits

In this way, one can rewrite (4.21) as follows:

$$\bar{I} = -\bar{Y}_M \bar{V} + \bar{I}_M \tag{4.23a}$$

$$\bar{I}_M = \bar{Y}_M \bar{E}' + \bar{I}_{saliency} \tag{4.23b}$$

$$\bar{I}_{saliency} = j\,\frac{1}{2}\left(\frac{x'_d - x'_q}{{r_a}^2 + x'_d\,x'_q}\right)\left[\left(\bar{E}'\right)^* - \left(\bar{V}\right)^*\right] e^{j2\delta} \tag{4.23c}$$

where all phasor quantities are given in the system reference frame and $\bar{E}' = \left(e'_q + j\,e'_d\right) e^{j\delta}$.

The equivalent circuit associated with the synchronous machine, represented by the algebraic equations (4.23), is depicted in the Figure 4.6

For the classical synchronous machine model, a further approximation is often assumed, which ignores the transient saliency of the machine, i.e., $x'_d = x'_q$. This way, the fictitious admittance $\bar{Y}_M$ becomes the inverse of the $r_a + j\,x'_d$, which corresponds to the transient impedance of the machine. Moreover, since flux linkages are also considered constant during the period of interest, the voltage $\bar{E}'$ also remains constant.

(d) *Initial Conditions*

The initial conditions for the differential equations of the machine, given in (4.13) and (4.16), are usually computed from a power flow solution, previously calculated for a specific operating condition of interest. Therefore, before starting a transient stability simulation, each synchronous machine is assumed to be in steady state, i.e., all derivatives are made equal to zero.

Since all generated power and terminal voltages in the system reference frame are available from the power flow solution, one can first make the derivatives in (4.16) zero and solve for

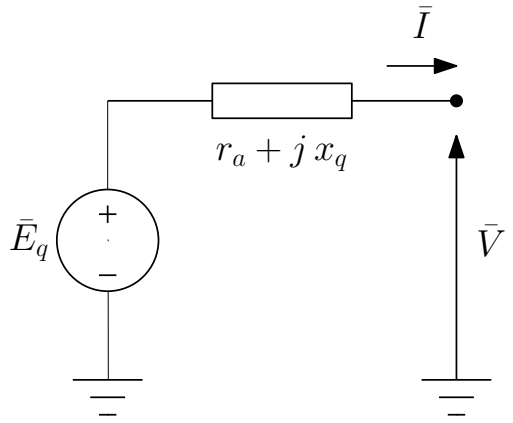


Figure 4.7. Steady-state equivalent circuit of the synchronous machine.

e'_q and e'_d, which yields (4.24).

$$e'_q = e_{fd} + (x_d - x'_d)\, i_d \tag{4.24a}$$

$$e'_d = -\left(x_q - x'_q\right) i_q \tag{4.24b}$$

Now, combining (4.24) with the stator algebraic equations (4.15) results (4.25).

$$v_q = -r_a i_q + x_d i_d + e_{fd} \tag{4.25a}$$

$$v_d = -r_a i_d - x_q i_q \tag{4.25b}$$

Rewriting then (4.25) using phasors in the system reference frame yields (4.26), which can be seen as voltage $\bar{E}_q$ behind the impedance $r_a + j\,x_q$, as depicted in Figure 4.7. From (4.26b), it can be observed that the direction of controlled voltage $\bar{E}_q$ coincides with the q-axis of the machine, as illustrated in Figure 4.8. Hence, the angle of $\bar{E}_q$ equals δ.

$$\bar{V} = -\left(r_a + j\,x_q\right)\bar{I} + \bar{E}_q \tag{4.26a}$$

$$\bar{E}_q = E_q\, e^{j\delta} \tag{4.26b}$$

$$E_q = e_{fd} + (x_d - x_q)\, i_d \tag{4.26c}$$

Since the machine terminal voltage $\bar{V}$ and the apparent power $\bar{S} = P + j\,Q$ generated by the machine are known from the power flow calculations, one can compute the stator current $\bar{I}$ as shown below.

$$\bar{I} = \left(\frac{P + j\,Q}{\bar{V}}\right)^* \tag{4.27}$$

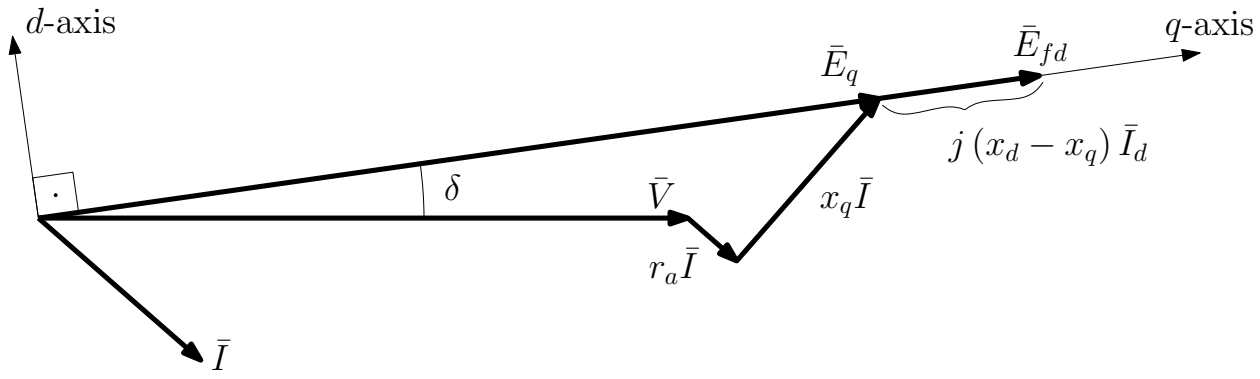


Figure 4.8. Steady-state phasor diagram of the synchronous machine, where $\bar{E}_{fd} = e_{fd}\, e^{j\delta}$ and $\bar{I}_d = j\, i_d\, e^{j\delta}$.

Now, plugging the previous $\bar{I}$ into (4.26a), one obtain $\bar{E}_q$, which, from (4.26b), gives the load angle δ. Once δ is known, both current components i_q and i_d can also be found as follows:

$$i_q = \Re\left\{\bar{I}\, e^{-j\delta}\right\} \tag{4.28a}$$

$$i_d = \Im\left\{\bar{I}\, e^{-j\delta}\right\} \tag{4.28b}$$

For the initial value of the excitation field voltage e_{fd}, (4.26c) is used along with the voltage $E_q = |\bar{E}_q|$ previously calculated from (4.26a). And, to finalize the initialization of the electrical differential equations, the transient voltages e'_q and e'_d are calculated from (4.24).

As for the steady-state of the mechanical equations (4.13), the frequency deviation ω must be set to zero, which guarantees that the load angle δ remains constant. Such a requirement is only met when the mechanical and electrical power match, i.e.,

$$P_m = P_e = e'_d\, i_d + e'_q\, i_q + \left(x'_d - x'_q\right)\, i_q\, i_d \tag{4.29}$$

4.2 Sequential Transient Stability Simulator

The reasons for implementing of a sequential transient stability simulator is twofold. First, it will serve as basis for the parallel version implementation, which is discussed in the next section. Second, the sequential simulator will provide a fair base for performance measurement of the parallel transient stability simulator, since both simulators share the same core functions and programming techniques.

The sequential transient stability simulator implemented complies with the alternating

solution method, explained in Section 4.1.1. The flow chart of the present implementation is given in Figure 4.9.

The preparation of a simulation starts with a parsing routine that reads the system data file and the contingency information. The system data files include information regarding generators and their associated prime movers and controllers, loads, and any other dynamic device present in the system. Subsequently, all data structures for dynamic devices, such as generators, and loads are allocated and initialized. Initial conditions for the specified operating condition are also computed for all dynamic devices connected to the system. Still during the pre-processing stage, the transmission system admittance matrix $\mathbf{Y}$ is formed, taking into consideration the models for transmission lines, transformers, loads and generators discussed in Section 4.1.2.

At this point the actual simulation starts. Before the computation of a time step proceeds, contingency statuses are checked, so that any due switching operation has to be performed. At the beginning of the simulation or whenever topology changes occur, the admittance matrix $\mathbf{Y}$ is factorized for later use during the network iterative solution. Non-integrable variables, such as voltages $\mathbf{v}$ at load and generation buses, are also predicted by extrapolation and history terms $\mathbf{x}_h$ associated with the state variables are computed. The formula used for extrapolating the voltages is shown in (4.30) (Stott, 1979). At the end of this stage, the equations (4.4) are ready for computation, which is performed next.

$$\bar{V}_{ext} = \frac{\bar{V}^2(t - \Delta t)}{\bar{V}(t - 2\Delta t)} \tag{4.30}$$

Now, the inner loop responsible for solving the nonlinear network starts with the iteration counter $m = 0$. At the beginning of the network solution, the currents injected into the passive network are computed for each dynamic and algebraic device, such as generators and polynomial loads. This procedure encompasses the integration of the dynamic models, denoted by (4.4a), and computation of the current injections $\mathbf{i}^m$ due to non-impedance loads, required by (4.4b). Afterwards, the voltages $\mathbf{v}^{m+1}$ across the system are solved from (4.4a), using the previously factorized $\mathbf{Y}$ matrix of the transmission network. With the first iterate $\mathbf{v}^{m+1}$, the dynamic equipments can update their state variables. If the difference between the initially predicted and updated voltages is negligible, i.e., $\|\mathbf{v}^{m+1} - \mathbf{v}^m\| \leq \varepsilon_v$, the simulation can move onto the next time step; otherwise, the iteration counter m is incremented by one, the current injections $\mathbf{i}^m$ are recalculated and the process repeats.

Test cases simulated with an implementation of the present algorithm will be discussed in Section 4.4.

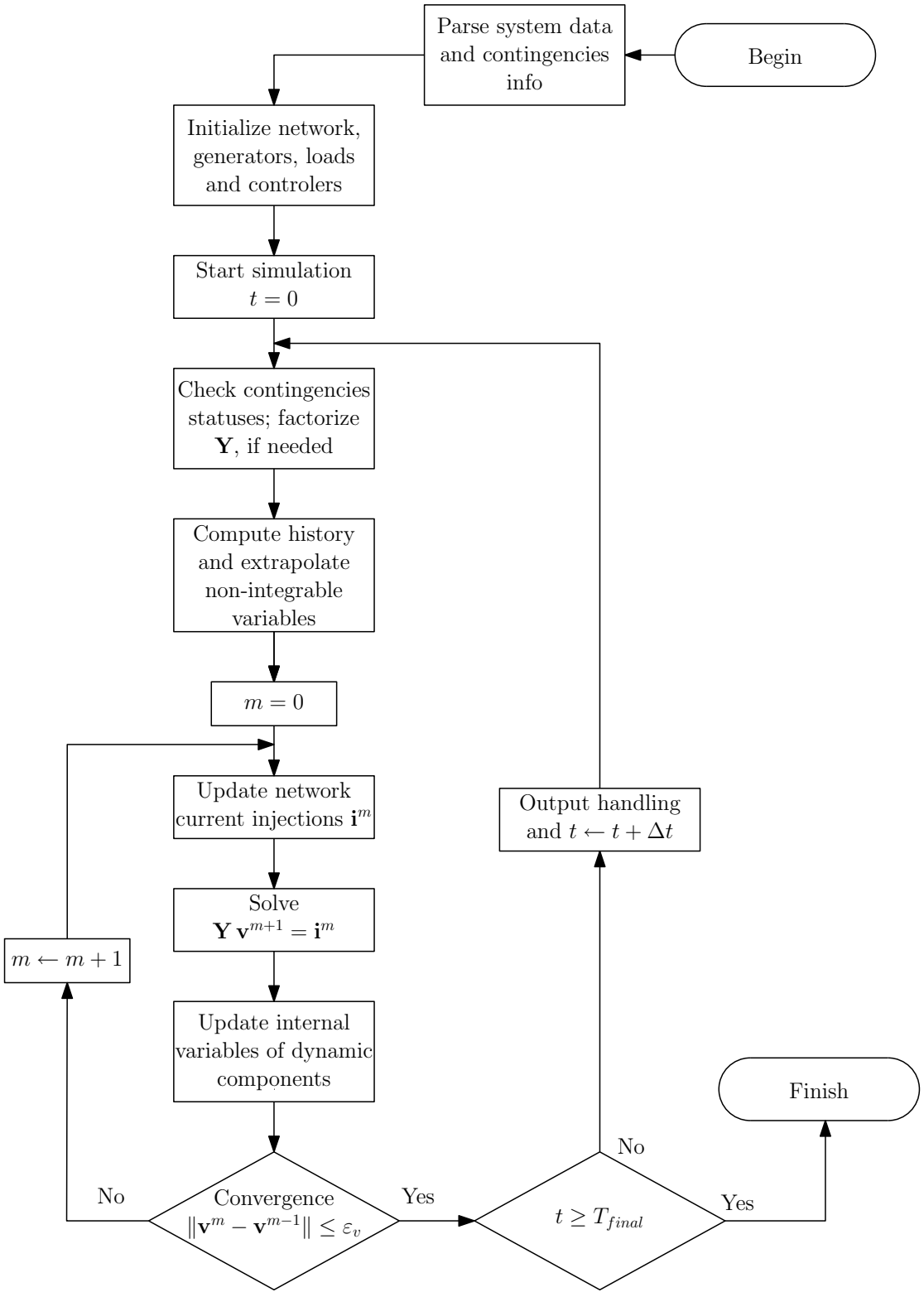


Figure 4.9. Flow chart of a transient stability program based on the partitioned approach.

4.3 MATE-based Parallel Transient Stability Simulator

A parallel transient stability simulator based on the network-based MATE algorithm will be discussed. The objective of such an application is to further assess the potential of the network-based MATE algorithm in improving existing transient stability simulators with the least programming effort and investment.

The present parallel transient stability simulator was targeted to run on distributed computing environments, such as computer clusters built with out-of-the-shelf PCs and network cards. Although distributed-memory was the system of choice, all ideas discussed next can be directly applied to shared-memory environments.

This implementation design combines the previously presented sequential transient stability simulator, as the basis of the code, and the network-based MATE algorithm, introduced in the Chapters 2 and 3, as the back-end of the sparse linear systems solver.

As such, the parallel alternating method for the transient stability solution is summarized by (4.31). In this set of equations, all variables are computed at time t and, hence, it is kept implicit, except for the history term $\mathbf{x}_{hk}(t)$, which depends on past computed values.

$$\mathbf{x}_k^m = \frac{\Delta t}{2}\,\mathbf{f}_k\big(\mathbf{x}_k^m, \mathbf{v}_k^m\big) + \mathbf{x}_{hk} \tag{4.31a}$$

$$\mathbf{Y}\,\mathbf{v}^{m+1} = \mathbf{i}\big(\mathbf{x}_k^m, \mathbf{v}_k^m\big) \tag{4.31b}$$

where $k = 1, \ldots, p$ represents the number of subsystems, which the original system is torn into, $m = 0, 1, \ldots$ is the iteration counter and, the history term $\mathbf{x}_{hk}(t)$ is given below.

$$\mathbf{x}_{hk}(t) = \mathbf{x}_k(t - \Delta t) + \frac{\Delta t}{2}\,\mathbf{f}_k\big(\mathbf{x}_k(t - \Delta t), \mathbf{v}_k(t - \Delta t)\big) \tag{4.32}$$

In (4.31), the state variables are presented in a partitioned manner, where each $\mathbf{x}_k^m$, assigned to the subsystem $\mathbb{S}_k$, is associated with only a portion of the state variables of the untorn system $\mathbf{x}^m$. And, since each subsystem contain only a set of the dynamic devices, originally connected to the untorn system, each set of discretized dynamic equations, given in (4.31a), only depends on the voltages $\mathbf{v}_k^m$ at buses belonging to the same subsystem $\mathbb{S}_k$.

Transient stability simulators can benefit considerably from the partitioning of the dynamic equations alone, based on the fact that most production-grade transient stability simulators usually spend from 60 to 80% of the simulation time integrating the dynamic models (Brasch et al., 1979; Wu et al., 1995).

The actual MATE-based parallel transient stability simulator can be split into three main stages: the partitioning stage, the pre-processing stage and solution stage. Each of these

stages are discussed in the sequence.

4.3.1 System Partitioning Stage

Similarly to the network-base MATE algorithm, the parallel transient stability simulator needs a system partitioning phase, where subsystems and interconnection links are topologically identified prior the simulation.

During the partitioning phase, the system under study in torn into a number subsystems, namely, p. Each of these subsystems will, therefore, contain a number of buses, branches and their associated dynamic devices. In turn, these subsystems are interconnected by the system of links.

Since actual power systems evolve incrementally, by addition of new buses or lines into the systems, their associated subsystems are expected to follow the same trend. Keeping this fact in mind, repartitioning of the systems is also expected to be mostly required in cases a different number of partitions are needed. Therefore, the approach adopted for this implementation was splitting the system data files, into several files required by the subsystems and link solver systems. As consequence, the partitioning routine does not need to be invoked in every simulation.

The subsystems files contain the data associated with their own transmission networks, generators, loads and local links. The link solver file, in turn, contain the global links information. Each link is defined by the following parameters:

Global ID: Identification of the link solver among the global links.

Local ID: Identification of the link solver among the local links.

From Subsystem ID: Subsystem, which this link is leaving

From Bus ID: Bus, which this link is leaving from.

To Subsystem ID: Subsystem, which this link is arriving.

To Bus ID: Bus, which this link is arriving to.

Orientation: 1 for orientation matching the one defined for the global link; -1, otherwise.

This set of parameters allow the assembling of the subsystem-to-border mappings $\mathbf{Q}_k$, defined in (2.8), and the link-to-border mappings $\mathbf{R}_k$, defined in (2.21), which, ultimately, provide interconnect all subsystems.

For the actual partitioning of the system, the METIS library (Karypis & Kumar, 1998a,b) was employed. This library provides a set of partitioning routines for undirected graphs

based on multilevel algorithms, such as the multilevel recursive bisection and multilevel k-way partitioning. Among the objective functions minimized during the partitioning phase was the minimum number of global links in the system.

The basic tasks in the system partitioning stage are:

(a) Parsing system data and initialize system structures.

(b) Undirected graph generation for the transmission network of the system under study. Graph data format is described in (Karypis & Kumar, 1998b).

(c) Undirected subgraphs generation for each generated, including local and global links, according to the partitioning produced by METIS.

(d) Subsystems construction based on the constructed subgraphs and original system data, such as generators and loads.

(e) Subsystems and link solver files generation.

4.3.2 Pre-processing Stage

In the pre-processing stage, the data structures required for storing and accessing the system data are allocated and initialized.

Following the network-based MATE algorithm idea, two types of processes are spawned at beginning of the simulation: subsystems and link solver processes. According to the flow chart presented in Figure 4.10, each processes start loading and parsing their appropriate data files. In addition to the system data associated with the transmission network, generators and loads, subsystems also require an extra file with the information regarding the event to be simulated, such as a fault followed by a line tripping.

During this stage, each subsystem $\mathbb{S}_k$ identifies its own set of border nodes $\mathbb{B}_k$. The border nodes contained in $\mathbb{B}_k$ are then used, along with the local links information, discussed in Section 4.3.1, to assemble the subsystem-to-border mappings $\mathbf{Q}_k$, defined in (2.8), and the link-to-border mappings $\mathbf{R}_k$, defined in (2.21).

Moreover, during the iterative network solution, the link-to-border mappings $\mathbf{R}_k$ are required by the link solver to form the multi-area Thévenin equivalent from the subsystems' multi-node Thévenin equivalents, as observed from (2.25) and (2.22). Therefore, all mappings $\mathbf{R}_k$ need to be gathered at the link solver prior the simulation. This will be further discussed in the next section.

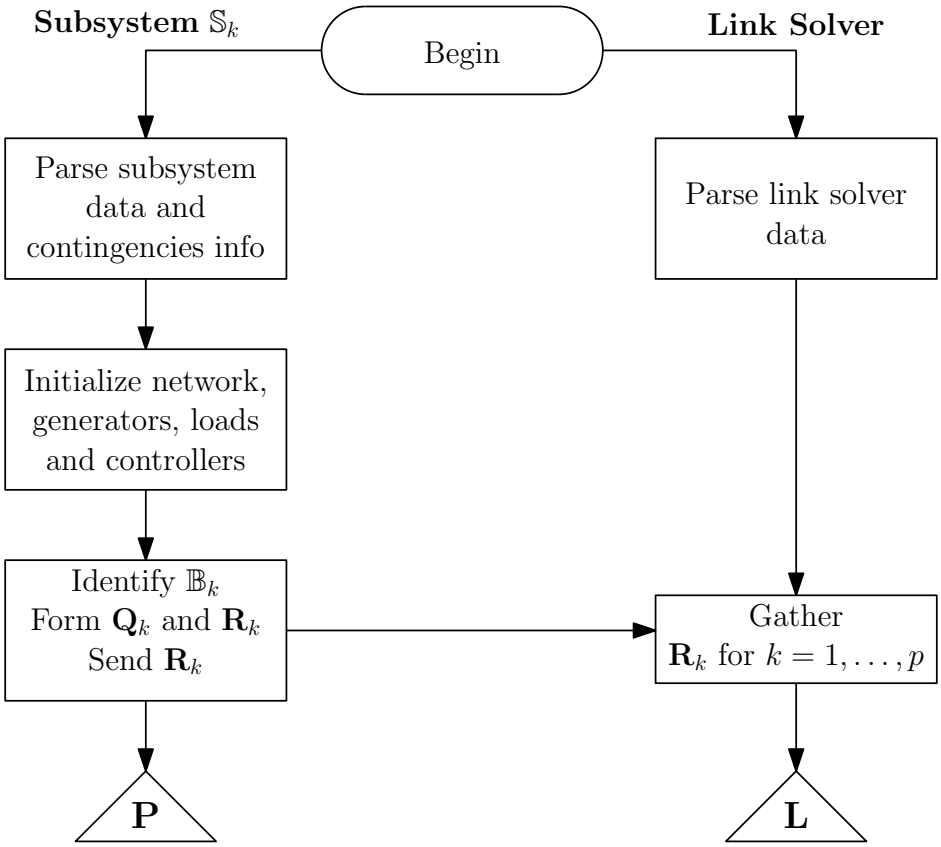


Figure 4.10. Flow chart of the pre-processing stage of the MATE-based parallel transient stability program (continues on Figure 4.11).

4.3.3 Solution Stage

The solution stage of the parallel transient stability follows closely the procedure discussed in Section 4.2, and is summarized in the flow chart presented in Figure 4.11. Therefore, the main differences between the two simulators will be analyzed next. These differences lie exactly at the points where interprocess communications are required, namely, the contingency statuses check, network-based MATE solver and convergence check.

Contingency Statuses Check

Although contingencies are usually applied within a subsystem and usually incur in the refactorization of the subsystem's local admittance matrix $\mathbf{Y}_k$, which, in turn, demands the link solver to rebuild and refactorize the multi-area Thévenin equivalent. Therefore, whenever a subsystem has its admittance matrix refactorized, the link solver needs to receive a signal, so it can proceed with the proper calculations.

Another aspect to be considered whenever any of the subsystems have to be refactorized

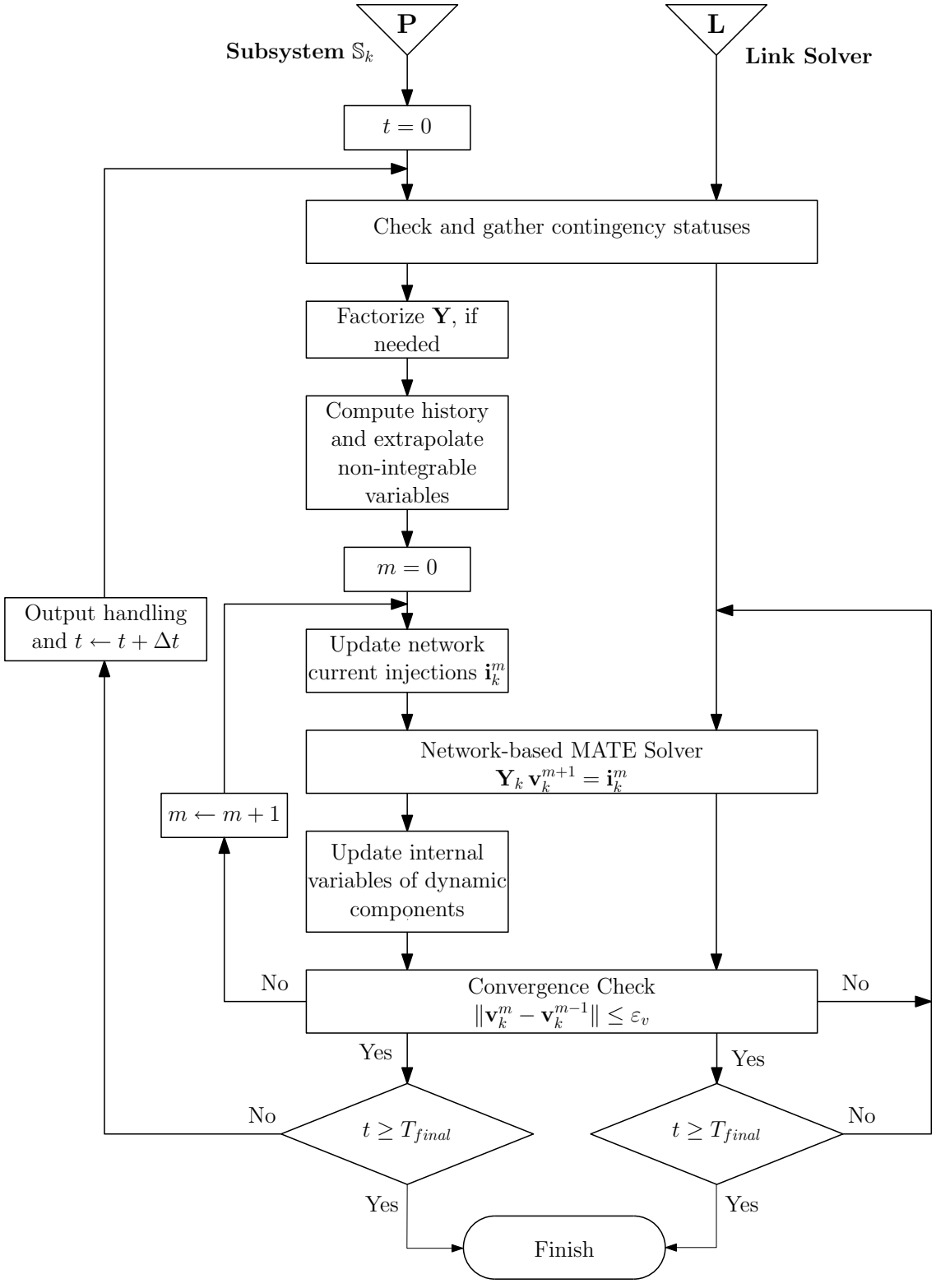


Figure 4.11. Flow chart of the solution stage of the MATE-based parallel transient stability program (continued from Figure 4.10).

is related with the manner MPI-1 standard (MPI Standard, 2008; Gropp et al., 1999) defines the interface for sending or receiving messages. First, one needs to keep in mind that the most basic MPI routines, the `MPI_SEND` for sending and the `MPI_RECV` for receiving messages, require, as passing arguments, the amount of data of a given type to be sent or received and which processes are participating in the communication.

Now, recall that the multi-node Thévenin equivalents, consisting of $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$, need to be sent from the subsystems to the link solver only when admittance matrices $\mathbf{Y}_k$ are (re)factorized, and only $\mathbf{e}_k^b$, otherwise. Hence, in the Thévenin equivalents gather (Section 3.2.3), the link solver has to keep track on not only the contingency statuses changes, but also which subsystems have changed.

Bearing these aspects in mind, the functionality of the contingency statuses check is described in Figure 4.12. In this adopted approach, each subsystem $\mathbb{S}_k$ sends to the link solver its own contingency status s_k, calculated according to (4.33a), by means of the `MPI_GATHER` routine. Afterwards, the link solver contingency status, s_0, is computed by finding the maximum of all gathered contingency statuses, as indicated in (4.33b). In this way, s_0 informs the link solver when a change occurred in the system, whereas s_k's identify the subsystems that suffered the change.

$$s_k = \begin{cases} 1 & \text{if } \mathbf{Y}_k \text{ was refactorized} \\ 0 & \text{otherwise} \end{cases} \tag{4.33a}$$

$$s_0 = \max_{k=1}^{p} c_k \tag{4.33b}$$

Network-based MATE Solver

In the present parallel transient stability simulator, the parallel network-based MATE solver (Chapters 2 and 3) is employed in the sparse linear solutions required by the iterative network computations (4.4b).

The flow chart, presented in Figure 4.13, summarizes the tasks performed by the network-based MATE solver.

At the beginning of a network solution, all subsystems $\mathbb{S}_k$ start computing their multi-node Thévenin equivalents, formed by $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$, according to (2.12), (2.16) and (2.17). As discussed previously, the recalculation of impedance matrix $\mathbf{Z}_k^b$ depends on the contingency status of the same subsystem, i.e., only when the admittance $\mathbf{Y}_k$ changes.

Once the Thévenin equivalents are computed, they can be gathered in the link solver process. This collective communication task depends on the contingency status of all subsystems. The contingency status are first checked on both subsystems and link solver processes.

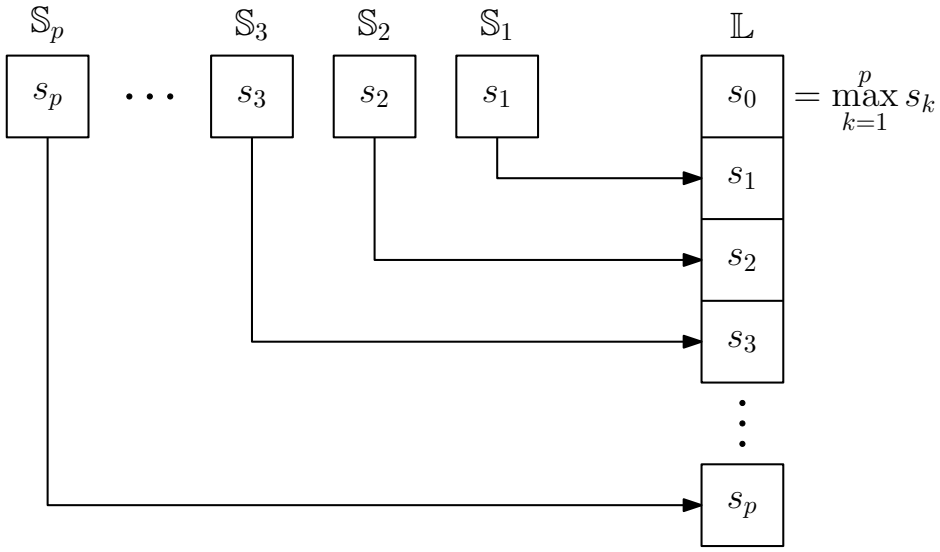


Figure 4.12. Contingency statuses check employed in the parallel MATE-based transient stability simulator.

In case changes in any $\mathbf{Y}_k$ are verified, the full Thévenin equivalents are sent(received) to(by) the link solver; otherwise, only the Thévenin voltages $\mathbf{e}_k^b$ are sent(received). After sending the Thévenin equivalents, the subsystems wait for the link currents to be calculated by the link solver.

After the link solver received the Thévenin equivalents, the multi-area Thévenin equivalent, formed by $\mathbf{Z}^l$ and $\mathbf{e}^l$, needs to be assembled according to the contingency statuses of the system. The assembling procedure is described by (2.22) and (2.25). However, if any of the contingency statuses are true, the full multi-area Thévenin needs to be rebuilt and the impedance matrix $\mathbf{Z}^l$ factorized; otherwise only the multi-area Thévenin voltages $\mathbf{e}^l$ are assembled. Afterwards, the link currents $\mathbf{i}^l$ can be computed from $\mathbf{Z}^l\,\mathbf{i}^l = \mathbf{e}^l$.

With the link currents $\mathbf{i}^l$ available, the link solver scatters them appropriately, i.e., respecting the link-to-border mappings $\mathbf{R}_k$, among the subsystems. The subsystems, in turn, that received their respective border node injection $\mathbf{i}_k^b$ due to the link currents $\mathbf{i}^l$, can, finally, update the local injections $\mathbf{i}_k^m$ and compute the local node voltages $\mathbf{v}_k$.

Convergence Check

At this point, all the nodal voltages $\mathbf{v}_k^{m+1}$ are solved and all the dynamic devices' state variables $\mathbf{x}_k^{m+1}$ updated, during the iteration m and for each subsystem $\mathbb{S}_k$. Now, the convergence of the solution needs to be checked against the predicted values of $\mathbf{v}_k^m$ and $\mathbf{x}_k^m$, which were used to computed the currents $\mathbf{i}_k^m$.

In the convergence check stage, the convergence of each subsystem $\mathbb{S}_k$ can be verified

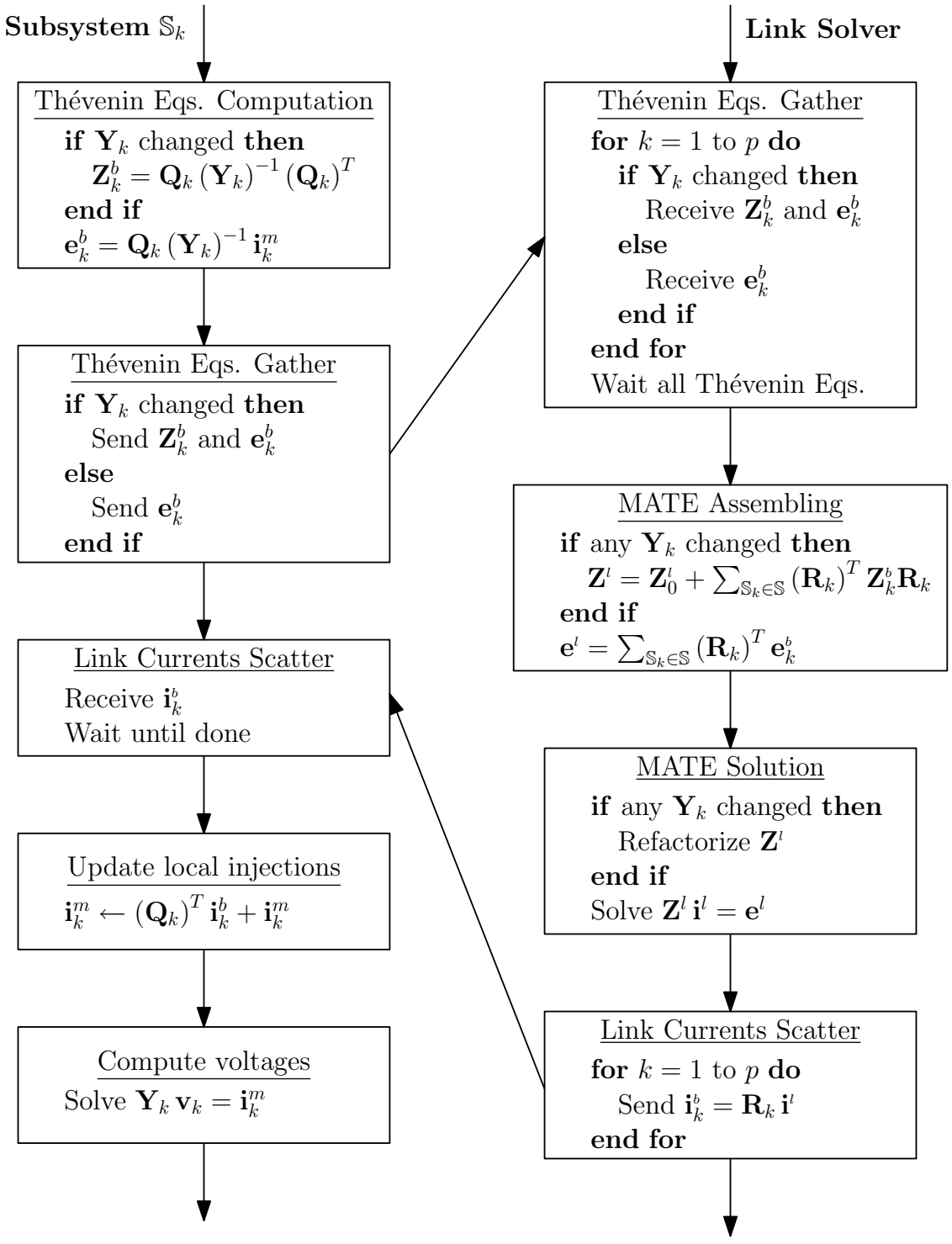


Figure 4.13. Network-based MATE parallel linear solver detail.

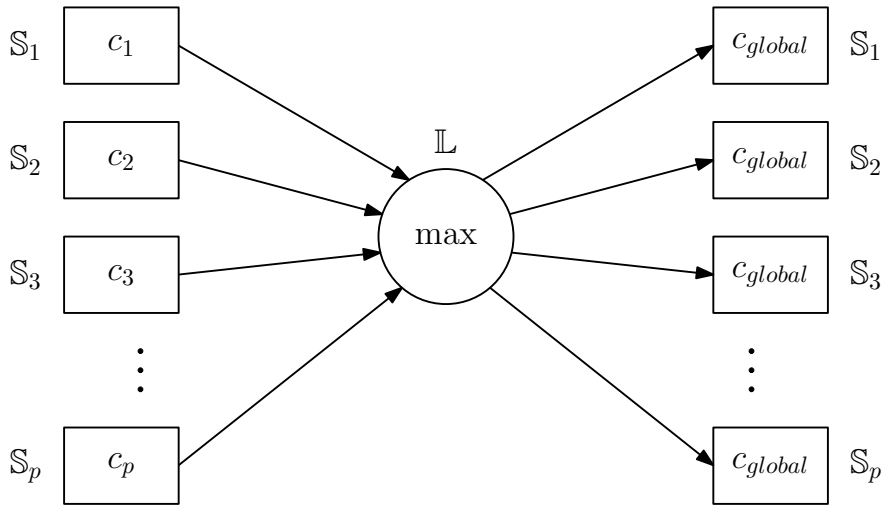


Figure 4.14. Convergence check employed in the parallel MATE-based transient stability simulator.

through the condition $\|\mathbf{v}_k^{m+1} - \mathbf{v}_k^m\| \leq \varepsilon_v$, where ε_v is the error tolerance applied to voltages. Although the convergence check can be performed by each subsystem independently, a global warning about the convergence status still needs to be issued to all subsystems and link solver, so all the processes know whether or not start a new iteration.

For this task, the convergence check is performed as illustrated in Figure 4.14. This procedure starts by each subsystem $\mathbb{S}_k$ calculating a convergence flag, as denoted by (4.34a), which is 1 if the system is not converged and 0 otherwise. Afterwards all local convergence flag are gathered in the link solver $\mathbb{L}$. In the link solver, a global convergence flag c_global is calculated, as in (4.34b), and broadcasted back to all subsystems. This operation can be performed by the collective `MPI_Allreduce` routine applied with the `MPI_MAX` operator (MPI Standard, 2008; Gropp et al., 1999).

$$c_k = \begin{cases} 0 & \text{if } \|\mathbf{v}_k^{m+1} - \mathbf{v}_k^m\| \leq \varepsilon_v \\ 1 & \text{otherwise} \end{cases} \tag{4.34a}$$

$$c_{global} = \max_{k=1}^{p} c_k \tag{4.34b}$$

Whether the program should keep track of the convergence of each subsystem individually is not clear, although it is possible. Different subsystems would perform more or less iterations under such conditions. A direct consequence of the distinct number of iterations would be a further unbalance in the computations in the subsystems and, hence, longer idle periods in the faster-convergent subsystems, which would have to wait for the slower-convergent subsystems to conclude the iterative process. However, further investigations are

Table 4.1. Summary of the SSBI system.

Element	Quantity
Buses	1916
Transmission Lines	2788
Generators	77
Loads	1212

needed concerning local convergence check, which goes beyond of the scope of the present work.

4.4 Performance Analysis

For analyzing the performance of the parallel transient stability simulator, described in Section 4.3, with respect to its sequential counterpart, discussed in Section 4.2, a reduced version of the South-Southeastern Brazilian Interconnected (thereafter, SSBI) System will be simulated. An overview of the Brazilian Interconnected System in its entirety is shown in Figure 4.15, whereas a summary of the SSBI system is given in Table 4.1.

In addition, two other aspects associated with the parallel transient stability simulation will be studied in this section: the parallelization of the network solution with the network-based MATE algorithm and the parallelization of the integration of the dynamic models. As discussed in Section 4.1, these are the major steps constituting the alternating solution approach for transient stability simulations.

4.4.1 South-Southeastern Brazilian Interconnected System Partitioning

Employing the system partitioning tool described in Section 4.3.1, the SSBI system was torn into 2 to up to 14 subsystems. A summary of the generated partitions is presented in Table 4.4.

Since the parallel transient stability simulator implementation relies on repeated solutions of the nodal equations and only a few factorizations due to topology changes, the network-based MATE algorithm performance for the present partitioning schemes can be qualitatively evaluated by means of the penalty factors defined in (3.31). Both link solver and communication penalty factors calculated from the partitioning information, given in Table 4.4, are shown in Figure 4.17.

Figure 4.15. Brazilian National Interconnected System. (`http://www.ons.org.br/conheca_sistema/mapas_sin.aspx#`)

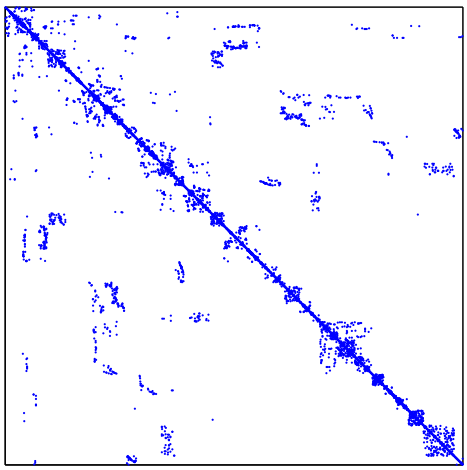


Figure 4.16. South-Southeastern Brazilian Interconnected System admittance matrix (1916 buses and 2788 branches).

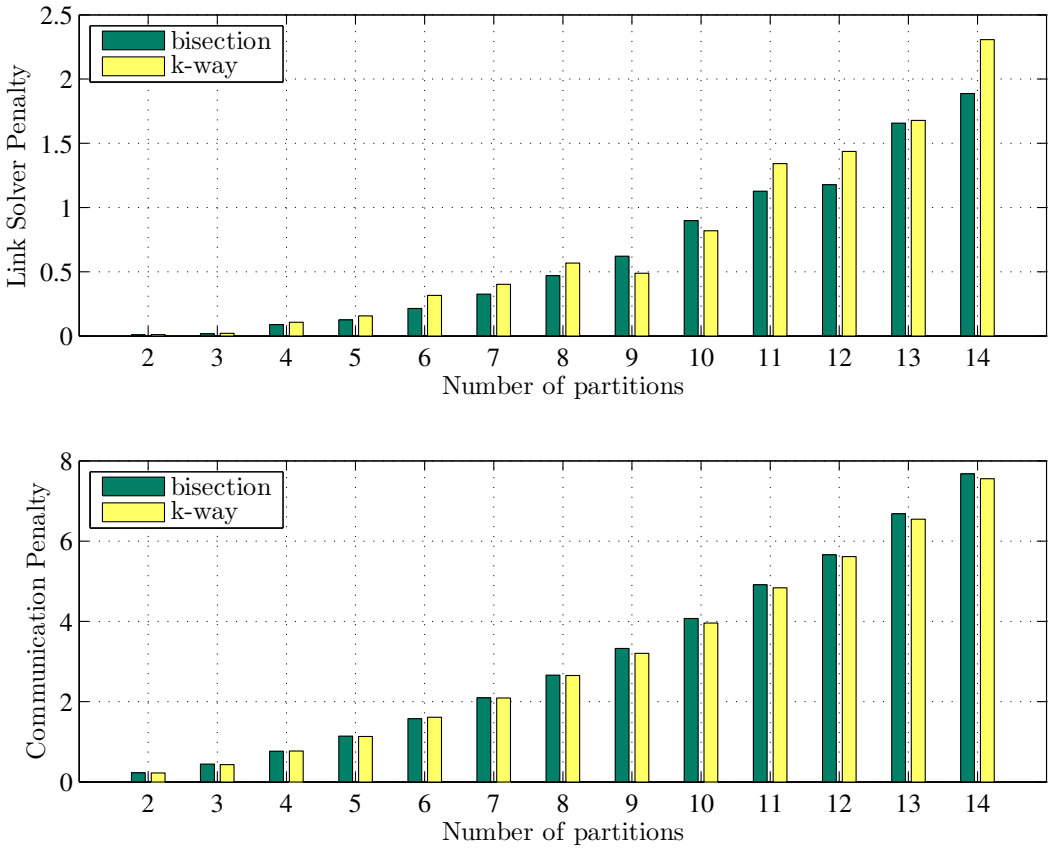


Figure 4.17. Link solver and communication penalty factors relative to the SSBI subsystems summarized in Table 4.4.

From these graphs, both penalty factors remain below unity only for only a very reduced number of partitions, regardless of the partitioning technique. Thus, a rapid degradation of the performance of the network-based MATE algorithm is expected as the number of partitions increases, because of the high penalty factors verified. In addition, the graphs also show that the communication burden represents the biggest bottleneck for the network-based MATE algorithm applied to the present partitions due to the high values of its associated penalty factor. Based on the fact that the penalty factors are normalized quantities with respect to the subsystems workload, one can further conclude that such performance degradation occurs due the reduced size of the subsystems in comparison with the link solver and communication.

4.4.2 Timings for the SSBI System

For the subsequent evaluations, a 100 ms short-circuit at the 765 kV Foz do Iguaçu substation, followed by the opening of one of the four circuits that connect Foz do Iguaçu and Itaipu

Bi-national substations was simulated for 10 s, with a time step of 4 ms. The SSBI system was modelled by means of classical synchronous machine models, in addition to constant power loads for voltages above 0.8 p.u. and constant impedance loads otherwise. For more information on the modelling of the system, see Section 4.1.2.

Timings for the sequential transient stability simulation

During the sequential transient stability simulation, the solution of 2,501 time steps was performed, which required 7,313 system solutions, i.e., iterations. The average number of iterations per time step was 2.92, with a standard deviation of 0.65, which conforms with the literature (Dommel & Sato, 1972; Stott, 1979). These statistics are summarized in Table 4.2. The timings recorded for the same simulation are given in Table 4.3. Ignoring the simulation setup and output handling, the network solution corresponded to about 25% of the computation time, while the dynamic models computations amounted to about 65% of the computation time. These proportions are also in agreement with the ones associated with production-grade transient stability simulators, as reported in the literature (Brasch et al., 1979; Wu et al., 1995).

Timings for the MATE-based parallel transient stability simulation

The same simulation of the SSBI system was performed using the parallel transient stability simulator described in Section 4.3. The timings recorded from the simulations, which ignore simulation setup and output handling time, are graphically shown in Figure 4.18.

These timings show that both multilevel recursive bisection and k-way partitioning techniques yield similar performance to the parallel simulator. The timings are drastically reduced from about 8 s to about 3 s, when number of subsystems varies from two to about eight, while saturation is observed for higher number of subsystems. This saturation is justified by the increase of the link solver computational burden and communication overhead, as predicted by the penalty factors, shown in Figure 4.17, which presented values lower than the unity for just a few subsystems. The convergence check timings also contributes to the communication overhead, which also increases with the number of subsystems, or, in this case, number of processors. The other timings, namely, network currents $\mathbf{i}_k$ computations, Thévenin equivalents $\mathbf{Z}_k^b$ and $\mathbf{e}_k^b$ computations, solution of subsystems' voltages $\mathbf{v}_k$ and update of dynamic models' internal variables, decrease with the number of partitions, due to their close relationship with the size of each subsystem. That is because solving smaller subsystems requires, in general, less computational effort.

The saturation of the performance of the parallel transient stability simulators can be

Table 4.2. Statistics of the SSBI system simulation.

Discrimination	**Value**
Simulation time	10 s
Time step	4 ms
Solution steps	2,501
System solutions	7,313
Average of solutions per time step	2.92
Standard deviation of solutions per time step	0.65

Table 4.3. Summary of the sequential transient stability simulation.

Task	**Timing [s]**	**Participation [%]**
Setup	0.857	5.88
Fault check	0.001	0.01
Convergence check	1.277	8.76
History terms and extrapolation	0.305	2.10
Current injections	8.063	55.3
First time factorization	0.009	0.06
Same pattern factorization	0.005	0.04
Voltages solution	3.290	22.6
Variables update	0.085	0.59
Output handling	0.677	4.65
Total	**14.57 [s]**	

further illustrated by the link solver timings, given in Figure 4.19. In these timings, the link currents $\mathbf{i}^l$ computation and scattering increase almost linearly with the number of subsystems. This aspect is denoted by the almost constant participation factors associated with each of the latter tasks, i.e., about 70 to 80% of communication and 20 to 30% of computation. These participation factors also reinforce the qualitative information embedded in the penalty factors (Figure 4.17), which predicted a much higher communication overhead than the link solver computations.

Performance metrics of the MATE-based parallel transient stability simulation

In addition to the performance analysis of the MATE-based parallel transient stability simulator, the MATE-based parallel network solutions, dynamic models computations and convergence and contingency check will be individually assessed for the simulations performed with the SSBI system. In this manner, the contributions of each parallel computation to the overall simulation can be evaluated.

Following the definitions of speedup and efficiency, introduced in Section 1.1.2, the performance metrics of the MATE-based parallel transient stability simulation are given below:

$$\mathcal{S}_{TSS} = \frac{T^p_{TSS}}{T^s_{TSS}} \qquad \mathcal{E}_{TSS} = \frac{S_{TSS}}{p+1} \tag{4.35a}$$

$$\mathcal{S}_{NET} = \frac{T^p_{NET}}{T^s_{NET}} \qquad \mathcal{E}_{NET} = \frac{S_{NET}}{p+1} \tag{4.35b}$$

$$\mathcal{S}_{DYN} = \frac{T^p_{DYN}}{T^s_{DYN}} \qquad \mathcal{E}_{DYN} = \frac{S_{DYN}}{p} \tag{4.35c}$$

$$\mathcal{S}_{CCC} = \frac{T^p_{CCC}}{T^s_{CCC}} \qquad \mathcal{E}_{CCC} = \frac{S_{CCC}}{p+1} \tag{4.35d}$$

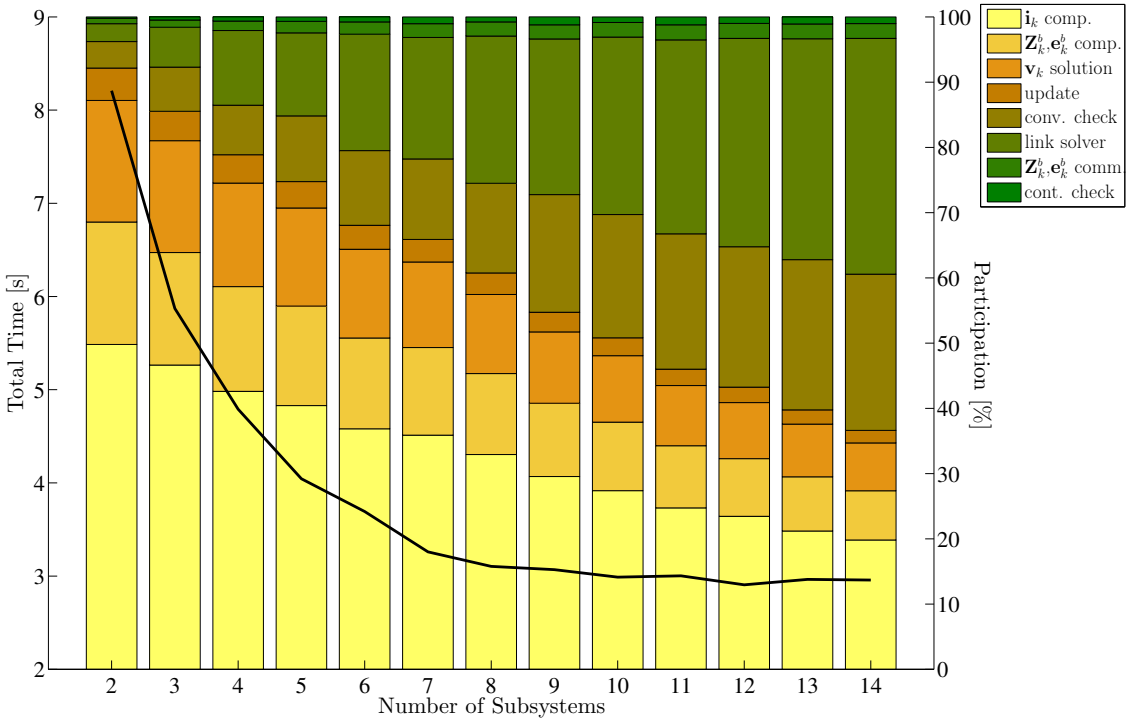
where T^s_Θ and T^p_Θ indicates whether the computation indicated by Θ is performed sequentially or in parallel, respectively, and $\mathcal{S}_\Theta$ and $\mathcal{E}_\Theta$ are the speedup and the efficiency of the parallel computations also indicated by Θ, which can be one of the following:

TSS = Transient stability simulation

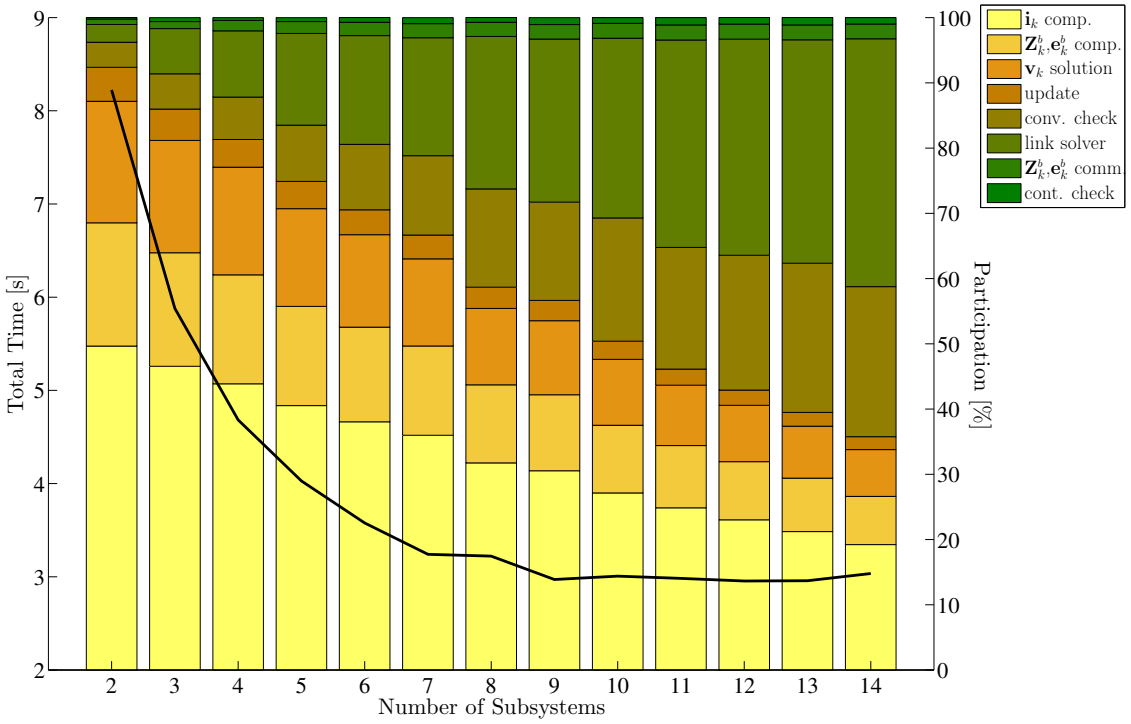
NET = Network solution

DYN = Computation of the dynamic models

CCC = Convergence and contingency status check

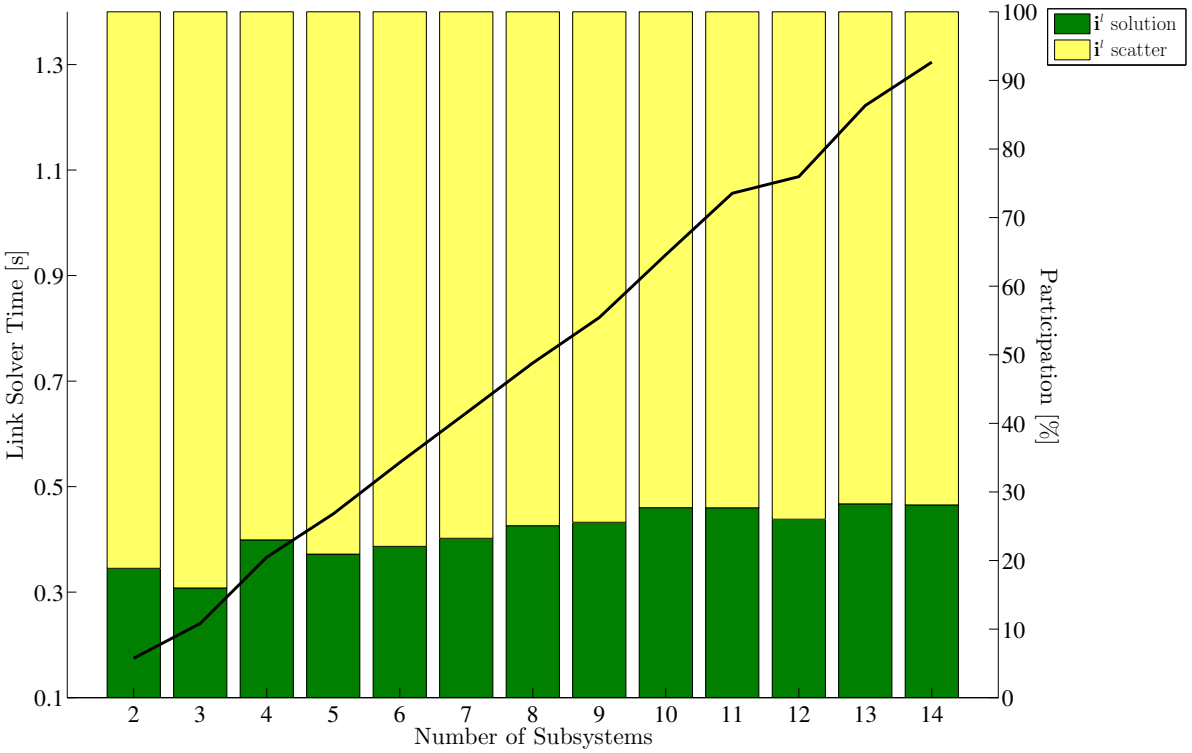


(a) Multilevel recursive bisection

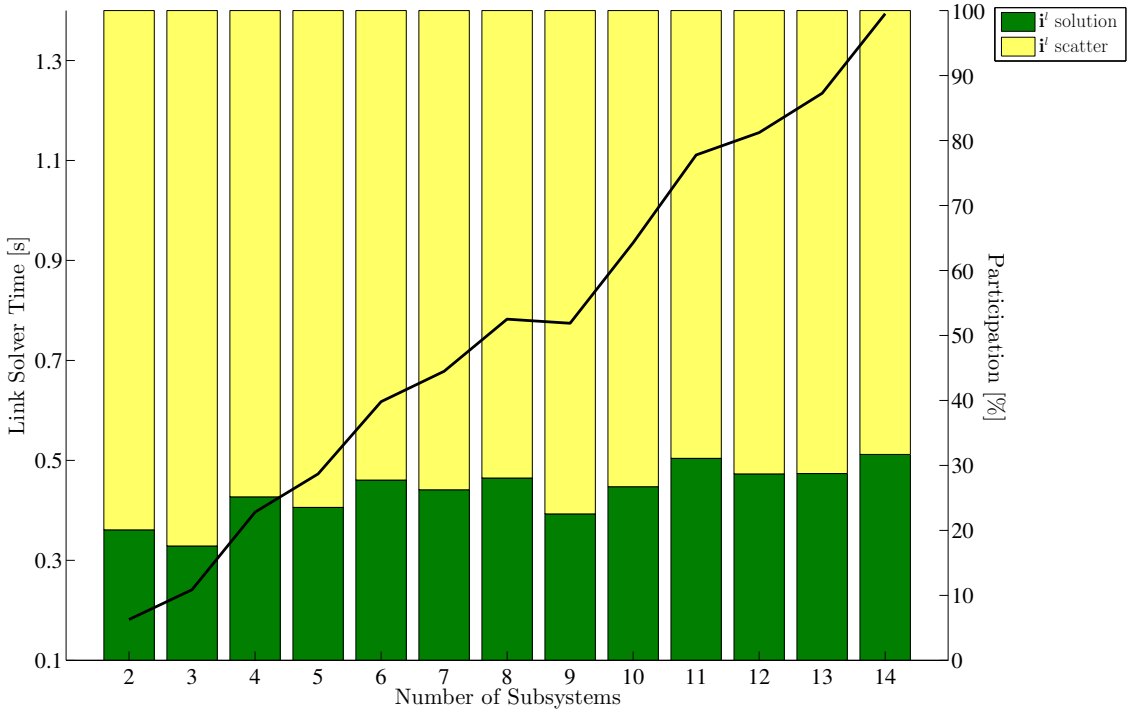


(b) Multilevel k-way partitioning

Figure 4.18. Timings of the parallel transient stability simulator for different partitioning heuristics.



(a) Multilevel recursive bisection



(b) Multilevel k-way partitioning

Figure 4.19. Timings of the link solver of the parallel transient stability simulator for different partitioning heuristics.

Moreover, the definition of the efficiency of the dynamic models' parallelization, $\mathcal{E}_{DYN}$, given in (4.35c), considers only p processes rather than $p+1$, as assumed for the other tasks. That is because the link solver does not take part in the computations of the dynamic models.

Based on the timings obtained from both sequential and parallel transient stability simulators, the performance metrics, defined in (4.35), were then calculated and plotted in Figure 4.21, as functions of the number of subsystems p.

Due the intrinsic parallel nature of the computations of the dynamic models, partitioning the latter task among distinct processors yielded speedups as high as six times with eight subsystems. From two to four processors, the efficiency even remains higher than 100%, due to speedups higher than p, which characterize superlinear speedups.

For the MATE-based parallel network solutions, the achieved speedups saturate around two times for about six subsystems. That is explained by the fact that, as p increases, the subsystems become small compared to the communication overhead and link solver computations, as predicted by the penalty factors previously discussed in Figure 4.17. The efficiency, however, presents values ranging from 30 to 40% for two to six processors, i.e., before the speedup saturation. Compared with other parallel sparse linear solvers reported in the literature (see Chapter 1), these values are still very competitive.

Differently from the previous two tasks, the convergence and contingency status checks present speedups that decrease with the number of subsystems p. That is because the computations associated with, mostly, the convergence check are much faster than the required communications. Even though the computations for the convergence check decrease with the number of subsystems p, the communication increase with $\mathcal{O}(p \log p)$ (Grama et al., 2003), which makes the speedups reduce with larger number of subsystems. The data fitting of the convergence and contingency status check time T^p_{CCC}, given in Figure 4.20, confirms the previous statement.

In order to further the understanding on the impact of the gains of the individual tasks on the overall parallel transient stability simulation, one can expand the speedup $\mathcal{S}_{TSS}$, given in (4.35a), in terms of the individual sequential and parallel timings as denoted in (4.36).

$$\mathcal{S}_{TSS} = \frac{T^p_{TSS}}{T^s_{TSS}} = \frac{T^p_{NET} + T^p_{DYN} + T^p_{CCC}}{T^s_{NET} + T^s_{DYN} + T^s_{CCC}} \tag{4.36}$$

Re-writting the above expression in terms of individual speedups defined in (4.35) yields:

$$\mathcal{S}_{TSS} = f^s_{NET}\,\mathcal{S}_{NET} + f^s_{DYN}\,\mathcal{S}_{DYN} + f^s_{CCC}\,\mathcal{S}_{CCC} \tag{4.37}$$

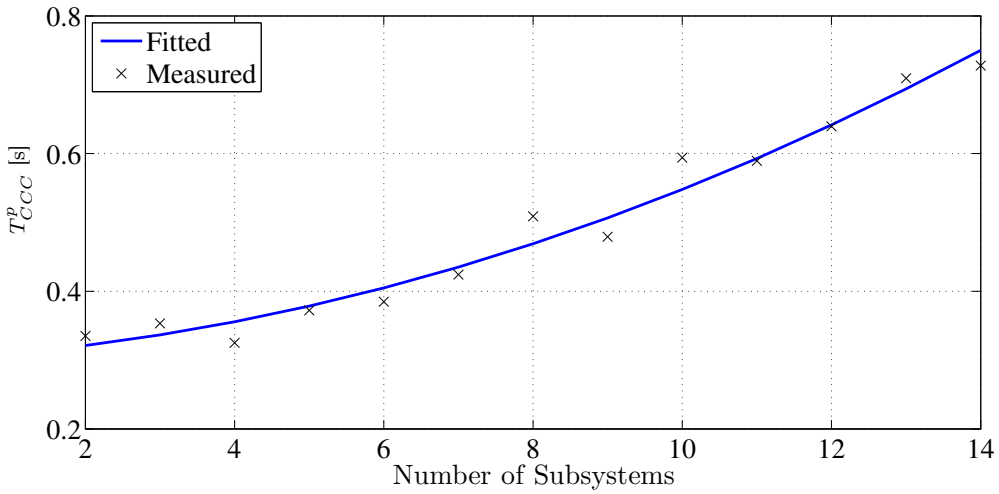


Figure 4.20. Convergence and contingency status check time T^p_{CCC} measured and fitted with $p \log p$ function.

where f^s_Θ stands for the fraction of the sequential parallel transient stability timing T^s_{TSS} that the task $\Theta = \{\text{NET}, \text{DYN}, \text{CCC}\}$ consumes, defined as follows:

$$f^s_\Theta = \frac{T^s_\Theta}{T^s_{TSS}} \tag{4.38}$$

In (4.37), the overall speedup of the transient stability simulation, $\mathcal{S}_{TSS}$, is expressed as a combination of the individual speedups of the parallel network solution, $\mathcal{S}_{NET}$, parallel solution of the dynamic and non-linear devices, $\mathcal{S}_{DYN}$, and the parallel convergence and contingency status check, $\mathcal{S}_{CCC}$. The foregoing speedups are weighted according to their corresponding task's proportion in the sequential transient stability simulation, which are given by the coefficients f^s_{NET}, f^s_{DYN} and f^s_{CCC}. Moreover, this relationship expresses, mathematically, that any speedup in the most time-consuming task of the transient stability simulation will have a much stronger impact on the overall computations performance.

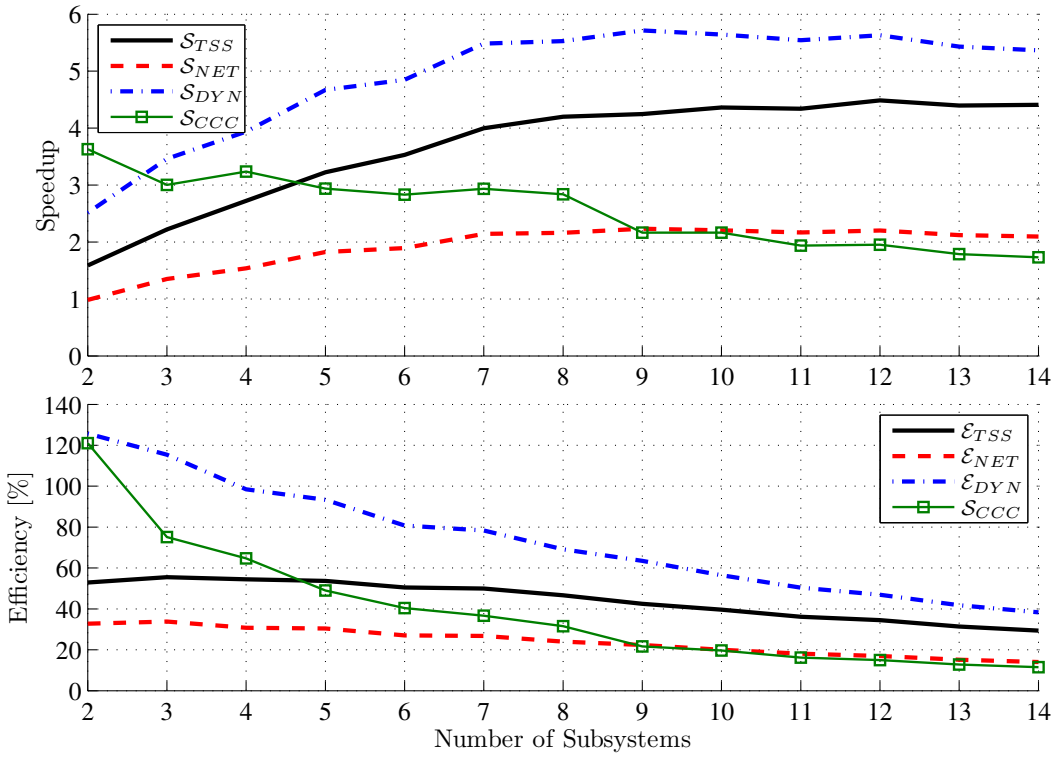
From the sequential transient stability timings, given in Table 4.3, one can obtain the proportions of each of the tasks, which results:

$$f^s_{NET} = 0.25 \qquad f^s_{DYN} = 0.65 \qquad f^s_{CCC} = 0.10$$

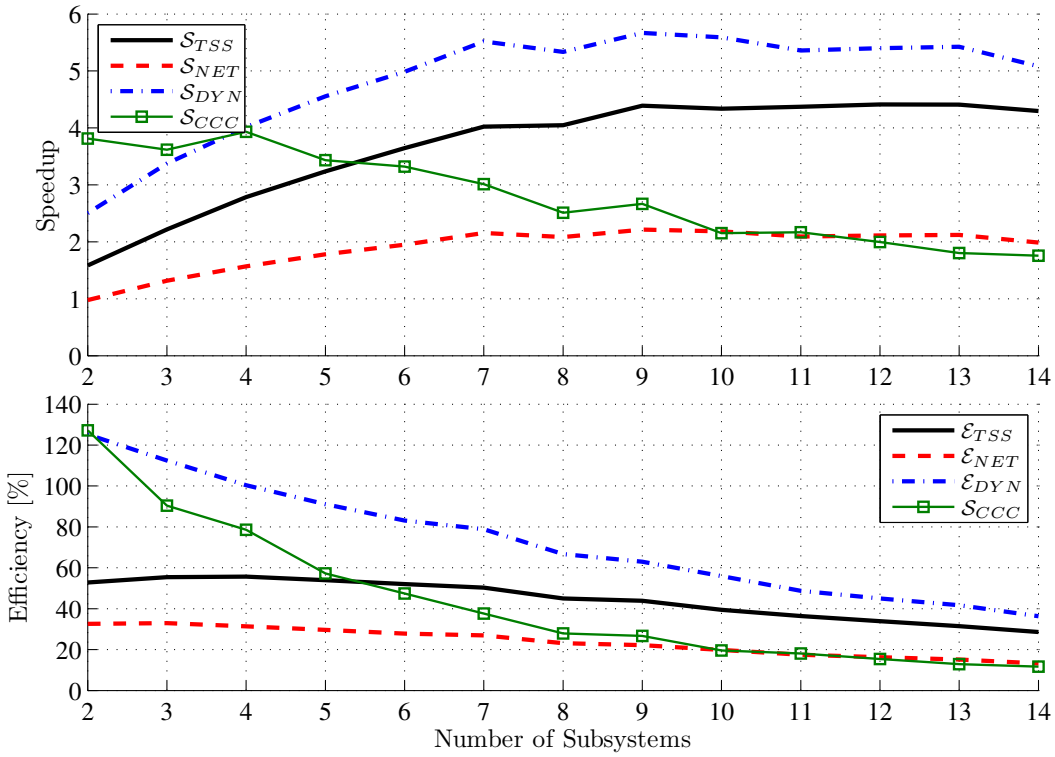
From Figure 4.21, the overall parallel transient stability solution presented a speedup of about 4.3 times, which approximates the solution of (4.37), with $S_{NET} = 2$, $S_{DYN} = 5.5$ and $S_{CCC} = 3$, calculated for 8 subsystems, and the above coefficients f^s_{NET}, f^s_{DYN} and f^s_{CCC}. These proportions also show that the speedups of parallel transient stability

simulation $\mathcal{S}_{TSS}$ are mostly influenced by the parallelization of the computations associated with the dynamic and non-linear devices, which occurs naturally when partitioning the system according to the network-based MATE algorithm. In the partitioning stage, however, only network topological information were taken into account, which causes imbalances in the subsystems in terms of dynamic and non-linear devices connected to the systems, as Table 4.4 shows. Such imbalances also influence the saturation observed in the speedups associated with the devices' computations, $\mathcal{S}_{DYN}$, and, consequently, in the speedup of the overall parallel transient stability simulation.

Although the network solutions correspond to only 25% of the sequential transient stability simulation, parallelizing the network solutions by means of the network-based MATE algorithm will definitely help speed up the transient stability solutions, specially for large systems. For instance, from Section 3.4.2, the WECC system (about 15,000 buses) was solved by the network-based MATE algorithm with a 6 times speedup with 14 subsystems. Now, assuming that the sequential tasks' proportions remain fixed, one may expect a direct contribution to the overall parallel transient stability simulation speedup of about 1.5 times from only the parallel network solutions.



(a) Multilevel recursive bisection algorithm.



(b) Multilevel k-way partitioning algorithm.

Figure 4.21. Performance metrics of the parallel transient stability simulator.

Table 4.4. Partitioning of the reduced South-Southwestern Brazilian Interconnected system using METIS library.

(a) Multi-level recursive bisection algorithm

p	l	$\mu(n_k)$	$\sigma(n_k)$	$\underline{n_k}$	$\overline{n_k}$	$\mu(b_k)$	$\sigma(b_k)$	$\underline{b_k}$	$\overline{b_k}$	$\mu(l_k)$	$\sigma(l_k)$	$\underline{l_k}$	$\overline{l_k}$	$\mu(g_k)$	$\sigma(g_k)$	$\underline{g_k}$	$\overline{g_k}$	$\mu(z_k)$	$\sigma(z_k)$	$\underline{z_k}$	$\overline{z_k}$
2	20	958.0	0.0	958	958	16.5	1.5	15	18	20.0	0.0	20	20	38.5	6.4	34	43	606.0	17.0	594	618
3	23	638.7	0.6	638	639	15.0	4.5	10	21	15.3	4.2	11	21	25.7	3.8	23	30	404.0	47.3	350	438
4	44	479.0	1.6	477	481	17.5	6.1	11	25	22.0	9.0	13	32	19.2	4.6	13	24	303.0	35.9	266	339
5	47	383.2	0.4	383	384	16.8	5.7	7	23	18.8	6.6	8	27	15.4	4.4	12	22	242.4	35.8	192	282
6	56	319.3	0.8	318	320	15.7	5.5	8	21	18.7	7.8	9	28	12.8	4.9	8	22	202.0	37.5	153	245
7	64	273.7	1.0	272	275	15.3	6.9	6	24	18.3	7.6	8	29	11.0	6.2	5	19	173.1	26.2	133	202
8	72	239.5	1.1	238	241	14.1	5.1	7	23	18.0	7.6	7	28	9.6	3.3	6	15	151.5	24.6	119	187
9	78	212.9	0.9	211	214	14.1	5.5	5	22	17.3	6.9	9	29	8.6	5.1	1	18	134.7	22.9	94	169
10	89	191.6	0.8	190	193	14.3	4.3	5	22	17.8	6.3	7	26	7.7	4.2	3	15	121.2	22.5	80	152
11	95	174.2	1.1	172	176	14.8	5.9	6	28	17.3	7.6	6	35	7.0	3.8	1	15	110.2	15.0	84	138
12	93	159.7	0.8	159	161	12.4	4.2	4	18	15.5	6.0	5	25	6.4	3.9	0	14	101.0	19.6	66	134
13	106	147.4	0.8	146	149	13.5	5.8	4	27	16.3	7.3	6	35	5.9	3.2	2	13	93.2	18.4	62	120
14	109	136.9	0.7	136	138	13.1	5.6	5	26	15.6	6.5	7	32	5.5	3.4	0	13	86.6	18.9	57	115

(b) Multi-level k-way algorithm

p	l	$\mu(n_k)$	$\sigma(n_k)$	$\underline{n_k}$	$\overline{n_k}$	$\mu(b_k)$	$\sigma(b_k)$	$\underline{b_k}$	$\overline{b_k}$	$\mu(l_k)$	$\sigma(l_k)$	$\underline{l_k}$	$\overline{l_k}$	$\mu(g_k)$	$\sigma(g_k)$	$\underline{g_k}$	$\overline{g_k}$	$\mu(z_k)$	$\sigma(z_k)$	$\underline{z_k}$	$\overline{z_k}$
2	22	958.0	33.9	934	982	18.0	1.0	17	19	22.0	0.0	22	22	38.5	0.7	38	39	606.0	19.8	592	620
3	25	638.7	19.1	621	659	14.0	3.6	11	19	16.7	4.9	11	23	25.7	0.6	25	26	404.0	44.0	355	440
4	49	479.0	12.8	467	492	20.8	10.6	9	36	24.5	11.9	10	39	19.2	5.7	14	26	303.0	18.6	281	325
5	53	383.2	9.1	371	392	18.4	7.2	7	26	21.2	8.7	7	32	15.4	3.0	11	19	242.4	30.5	190	269
6	69	319.3	7.6	309	328	20.0	5.8	12	26	23.0	6.8	14	31	12.8	6.4	8	25	202.0	28.4	154	241
7	72	273.7	6.7	265	282	17.4	8.4	6	32	20.6	10.3	8	39	11.0	5.6	5	21	173.1	18.8	146	194
8	80	239.5	6.3	230	246	16.1	7.3	6	27	20.0	8.6	9	33	9.6	4.5	3	17	151.5	26.6	112	189
9	70	212.9	5.9	200	219	13.0	5.2	5	19	15.6	6.5	6	27	8.6	4.7	3	15	134.7	25.6	93	172
10	86	191.6	4.6	185	198	14.0	5.2	4	21	17.2	7.4	5	30	7.7	4.7	4	17	121.2	19.3	93	158
11	105	174.2	6.6	157	179	15.6	6.9	4	28	19.1	10.1	4	37	7.0	3.8	3	15	110.2	17.2	86	139
12	104	159.7	6.7	141	165	14.0	6.5	3	24	17.3	8.7	3	32	6.4	4.5	2	18	101.0	19.3	74	125
13	108	147.4	3.5	140	151	13.9	6.7	3	24	16.6	8.8	5	33	5.9	5.2	0	16	93.2	18.8	59	127
14	122	136.9	6.3	118	144	13.9	7.3	4	31	17.4	9.7	6	42	5.5	3.1	1	11	86.6	14.7	59	109

- $\overline{x_k}$ means the maximum value of x_k
- $\underline{x_k}$ means the minimum value of x_k
- $\mu(x_k)$ means the average value of x_k
- $\sigma(x_k)$ means the standard deviation of x_k
- g_k represents the number of generators
- z_k represents the number of loads

4.5 Conclusion

A transient stability simulator based on the network-based MATE algorithm was discussed and implemented in its sequential and parallel versions. The solution approach adopted for the transient stability problem was the alternating solution method, where differential equations and network equations are solved in an alternate fashion. Basic models for the most elementary power systems devices, such as generators, transmission lines, transformers and composite loads, were also presented. A reduced version of the Brazilian Interconnected System, with about 2,000 buses, was employed in the subsequent evaluations.

The timings of the implemented sequential transient stability simulator presented a profile similar to those observed in industrial-grade programs: 65% of the computations associated with dynamic and non-linear devices, and 25% related to network solutions.

The performance metrics of the parallel transient stability simulator with respect to its sequential counterpart showed that the parallel network solutions and the distribution of the dynamic and non-linear devices among subsystems are fundamental in shaping the overall parallel transient stability simulation performance. For the simulated system, the parallel computation of the dynamic and non-linear elements achieved speedups as high as six with eight subsystems, while the parallel network solutions based on the network-based MATE algorithm achieved speedups of up to two times for the same number of subsystems. As a consequence, the overall parallel transient stability solution presented a speedup of approximately 4.5 times.

Chapter 5

Conclusion

In this thesis, the network-based *Multi-Area Thévenin Equivalent* (MATE) algorithm has been proposed and employed in the solution of the transient stability problem in a distributed parallel computing architecture. Theoretical performance analysis of the network-based MATE has also been developed. The resultant performance model provides qualitative and quantitative performance measures of the various stages of the algorithm when applied to the solution of a specific network on a given hardware/software setup. A large power system network, the WECC system with 14,327 buses was solved with the developed network-based MATE algorithm. A speedup of 6 times over conventional sparsity-oriented solutions was obtained with a 14-PC commodity cluster.

A parallel transient stability program was also developed, which employed the previous network-based MATE algorithm as the backend for the sparse linear solutions. The alternating algorithm for transient stability simulations was implemented on the top of the practically intact structure of the network-based MATE algorithm, which provided straight concurrent solution of groups of dynamic elements. Tests on the SSBI systems with 1916 buses revealed speedups of up to 4.3 times over a sequential version of the same transient stability solution algorithm.

The main conclusions of the present work will be summarized in sequence, along with some possible further investigations and final remarks about the MATE algorithm applied to the solution of bulk power systems.

5.1 Summary of Contributions

The main contributions of this thesis are:

1. **Introduction of the network-based MATE algorithm, which further optimizes the matrix-based MATE algorithm in terms of computation and communication overhead**

   The network-based MATE introduces new concepts, such as border nodes, multi-node and link Thévenin equivalents and their associated mappings (subsystem-to-border and link-to-border). Such concepts were proven to be helpful in reducing the computations

performed on the subsystems, as well as the communications required to form the link system in the link solver.

2. **Implementation of the network-based MATE algorithm on a commodity cluster employing ready-to-use sparsity and communication libraries**

   The network-based MATE algorithm was implemented on a PC cluster, consisting of several single-core PCs interfaced by a dedicated high-speed network. In order to minimize the code development time, generic and ready-to-use libraries, which implement all computational and communication kernels required by the network-based MATE, were employed. Sparse and dense linear solutions were provided by SuperLU (Li et al., 2003; Demmel et al., 1999) and GotoBLAS (Goto, 2006), respectively. More specifically, the GotoBLAS library provides implementations of BLAS (Basic Linear Algebra Subroutines) (Blackford et al., 2002) and LAPACK (Linear Algebra Package) (Anderson et al., 1999), which define a comprehensive standard interface for linear algebra routines. As for the interprocess communications, the *Message Passing Interface* (MPI) standard (MPI Standard, 2008) was adopted, whose implementation was provided by NMPI library (NICEVT, 2005), which is a version of the well known MPICH2 library (Argonne National Laboratory, 2007) over the *Scalable Coherent Interface* (SCI) (IEEE, 1993). Because all the aforementioned libraries are implemented in accordance to widely accepted programming standards, employing such libraries not only minimized the development time but also enhanced the portability of the code, which can be easily compiled for many parallel computing architectures ranging from PC clusters to supercomputers.

3. **Development of a performance model for the network-based MATE algorithm**

   Developing a performance model for the network-based MATE algorithm enabled the establishment of a theoretical speedup limit for the method, with respect to traditional sequential sparsity-oriented sparse linear solvers. In addition, the developed performance model helps evaluating the performance of the proposed algorithm for a given hardware/software setup and a specific power system network to be solved. This is a key piece of information that also helps one making decisions regarding the best suitable algorithm and computational environment for solving a given problem.

4. **Application of the parallel network-based MATE algorithm for the solution of the network equations associated with the transient stability simulation**

The proposed network-based MATE algorithm was employed in the solution of sparse linear systems, common in transient stability programs. For the 14,327-bus system, extracted from the North American Western Electricity Coordinating Council (WECC), speedups closely followed the theoretical speedup of $\frac{p}{2}$, where p is the number of partitions, reaching about 6 times with 14 partitions (Figure 3.23).

5. **Implementation of a parallel transient stability simulator based on the parallel network-based MATE algorithm**

   The network-based MATE algorithm was employed as the backend for the sparse linear solutions required by transient stability simulations. The network-based MATE algorithm not only provided an alternative for solving the network equation in parallel, but also lent a structure suitable for solving groups of differential equations associated with dynamic elements concurrently. Because the integration of the differential equations is the most time consuming task in most of the industrial-grade transient stability simulators (about 80% of the computational time), its parallelization becomes essential to yield maximum performance to any parallel transient stability simulator. Employing the alternating algorithm for solving the transient stability problem, tests on the SSBI systems with 1916 buses revealed speedups of up to 4.3 times over a sequential version of the same transient stability solution algorithm.

5.2 Future Work

The network-based MATE algorithm has been proved to be competitive with other parallel algorithms for the solution of large linear systems in transient stability simulations using commodity off-the-shelf PC clusters. However, there are still many related aspects suitable for further investigation, such as:

1. **Improved solution of the subsystems equations**

   As shown in the thesis, the theoretical speedup limit of the network-based MATE algorithm is $\frac{p}{2}$, where p is the number of partitions. This limit can be roughly explained by the fact that the subsystems need to be solved, at least, twice during each solution. Therefore, further optimizing the Thévenin equivalents computations may significantly increase the the foregoing speedup limit. One technique that can be employed for such an optimization is the sparse vector methods (Tinney et al., 1985), which can potentially reduce the number of non-zeros handled during the solution of systems with only a few injections. In this case, the theoretical speedup limit would approximate $\frac{p}{1+f}$,

where $f \in (0, 1]$ represents the computational factor due to the Thévenin equivalent solutions. For instance, as reported by Tinney et al. (1985), sparse vector methods achieved up to 20 times speedup over regular sparse forward and backward substitutions. If sparse vector methods were used to compute the Thévenin equivalent computations, the computational factor $f = \frac{1}{20}$ would yield a speedup limit of approximately $\frac{20}{21}p \approx 0.95\,p$, which means a practically linear speedup[10].

2. **Implementation of the MATE algorithm in more advanced parallel architectures**

   In the present work, each subsystem and the the system of equations associated with the interconnections among subsystems, i.e., link currents equations, were concurrently solved on single processors. Further acceleration of the solutions could be achieved by means of more advance parallel architectures, like multi-core processors and graphical processing units (GPU). With the tightly-coupled multiprocessing architectures provided by off-the-shelf multi-core processors, the MATE algorithm could be used for the coarse grain parallelization, i.e., defining subsystems and links; whereas finer grain parallelism (Huang & Wing, 1979; Wing & Huang, 1980; Wu & Bose, 1995; Amestoy & Duff, 2000; Armstrong et al., 2006) could be further exploited inside each subsystem or link solver. As such, each subsystem and the link solver could be mapped onto different multi-processing units and solved locally in parallel. And, as long as the interprocessor communications, i.e., number of links, are minimized at the MATE algorithm level, one may expect better scalability of the solutions on SMP machines. Yet another piece of hardware that has a great potential in accelerating floating-point operatins is the GPU. Differently from what has been observed for the CPU technology, the performance of GPUs is quickly accelerating, due to the inherent parallel nature of graphic computations. GPUs can achieve much higher arithmetic intensity with the same transistor count as regular CPUs (Owens et al., 2007).

3. **Performance analysis of the multi-level MATE applied to the solution of large power systems**

   Armstrong et al. (2006) shows that multi-level MATE is capable of considerably improving the performance of the MATE algorithm when only dense matrices are handled on a single processor. Due to the relatively reduced size of the systems solved by the multi-level MATE, sparsity-oriented techniques were not considered. Hence, it seems

[10]The theoretical speedup limit assumes both communications and link solver computations negligible, while only forward and backward substitutions are performed.

worthwhile developing a performance model for a sparsity-oriented version of multi-level MATE in order to determine possible performance gains and bottlenecks when applied to solving large power systems.

4. **Development of partitioning algorithms specially tailored for meeting both computational and communication requirements imposed by the MATE algorithm**

   Two different generic graph partitioning algorithms, namely, the multi-level recursive bisection and k-way algorithms (Karypis & Kumar, 1998b,a), were employed in this work for defining the subsystems solved by the MATE algorithm. From the results, the subsystems generated by both aforementioned partitioning heuristics yielded similar performance to the parallel forward and backward solutions implemented through the MATE algorithm. For the factorization, however, the load unbalance among the subsystems was evident, due to the uneven number of border nodes in each subsystem. As for the parallel transient stability program, computational costs involved in solving equations associated with generators and loads were not considered in the partitioning phase. Therefore, the MATE algorithm would certainly benefit from the development of a partitioning heuristic capable of balancing the load among many subsystems, considering the number of border nodes and computational costs of generators and loads in each subsystem, while keeping the number of interconnections among the subsystems low.

5. **Feasibility of geographically distributed power systems simulations based on the MATE algorithm**

   As pointed out by Wang & Morison (2006), another challenging issue is the modeling of the external system not observable by the SE (state estimator). Inclusion of these models in the real-time system models may require the development of adequate offline equivalent models, which can then be merged with the real-time SE models.

   Similarly to the method proposed by (Esmaeili & Kouhsari, 2007), the network-based MATE algorithm can also aid the deployment of geographically distributed transient stability simulations. In the MATE context, areas managed by distinct utilities can be represented by separate subsystems, whereas the tie lines interconnecting these areas are the links. In this manner, each utility can preserve the locality and the privacy of its own data, and needs only to make their system equivalents available, for the sake of the whole system's solution.

5.3 Final Remarks

The MATE algorithm, originally proposed by (Martí et al., 2002), has been reformulated from an electric network point of view, which yielded the network-based MATE algorithm introduced in this work. It has been shown that inherent characteristics of the electrical networks, revealed by the present development, can reduce both computational effort and communication overhead of a parallel implementation of the MATE algorithm.

Results of previously related researches and the present implemention of the MATE algorithm on a commodity cluster show that, in order to keep parallel computations efficient, one has always to keep in mind the nature of the problem to be solved and the target computing architecture. Some parallel algorithms may be more suitable to certain architectures than others.

For example, SuperLU DIST, whose goal is scalability when solving extremely large-scale unsymmetrical problems on massive distributed-memory supercomputers, may not perform well on a commodity cluster solving systems of moderate size. For the sake of comparison, timings for the parallel network-based MATE and SuperLU DIST when solving the two power systems studied in Section 3.3 and Section 4.4 on a commodity cluster are shown in Table 5.1. As can be observed, the network-based MATE scales well up to 8 processors, while the SuperLU DIST suffers from its high communication overhead in comparison with the computational load of each processing node. The timings show, for instance, that the network-based MATE algorithm was able to solve the 14,327-bus WECC system up to 26 times faster than SuperLU DIST.

In summary, the network-based MATE algorithm can be very efficient for solving sparse linear systems, commonly found in transient stability programs using commodity PC clusters built with out-of-the-shelf processors.

Table 5.1. Timings for 1 factorization and 1000 repeated solutions using the network-based MATE and SuperLU DIST on a 16 AMD Athlon^{TM} 64 2.5GHz processors cluster interconnected by a SCI-based network.

Processors	Solver	SSBI† (1,916 buses)	WECC† (14,327 buses)
1	MATE	-	-
	SuperLU Seq.	0.43	4.6
2	MATE	-	-
	SuperLU DIST	4.52	37.6
3	MATE	0.45	4.3
	SuperLU DIST	5.24	39.3
4	MATE	0.32	2.6
	SuperLU DIST	5.25	38.4
5	MATE	0.26	1.8
	SuperLU DIST	4.87	36.0
6	MATE	0.24	1.4
	SuperLU DIST	4.58	32.6
7	MATE	0.22	1.2
	SuperLU DIST	4.48	30.3
8	MATE	0.21	1.1
	SuperLU DIST	4.19	N/A‡

† All timings are given in [s]
‡ N/A means that the program returned an unknown error

Bibliography

Alexandrov, A., Ionescu, M. F., Schauser, K. E., & Scheiman, C. (1995). *LogGP: Incorporating Long Messages into the LogP Model — One step closer towards a realistic model for parallel computation.* Technical report, University of California at Santa Barbara.

Aloisio, G., Bochicchio, M., Scala, M. L., & Sbrizzai, R. (1997). A distributed computing approach for real-time transient stability analysis. *Power Systems, IEEE Transactions on*, 12(2), 981–987.

Alvarado, F. (1976). Computational complexity in power systems. *IEEE Trans. Power App. Syst.*, 95(4), 1028–1037.

Alvarado, F. (1979). Parallel solution of transient problems by trapezoidal integration. *IEEE Trans. Power App. Syst.*, PAS-98(3), 1080–1090.

Alvarado, F., Reitan, D., & Bahari-Kashani, M. (1977). Sparsity in diakoptic algorithms. *IEEE Trans. Power App. Syst.*, 96(5), 1450–1459.

Alvarado, F., Yu, D., & Betancourt, R. (1990). Partitioned sparse a^{-1} methods. *IEEE Trans. Power Syst.*, 5(2), 452–459.

Amdahl, G. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings* (pp. 483–485).

Amestoy, P. R. & Duff, I. S. (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng*, 184, 501—520.

Amestoy, P. R., Duff, I. S., L'excellent, J., & Li, X. S. (2001). Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Trans. Math. Softw.*, 27(4), 388–421.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., & Sorensen, D. (1999). *LAPACK Users' Guide.* Philadelphia, PA: Society for Industrial and Applied Mathematics, third edition.

Anderson, P. M., Fouad, A. A., of Electrical, I., Engineers, E., & Paul M. Anderson, A. A. F. (2003). *Power system control and stability*.

Andersson, G., Donalek, P., Farmer, R., Hatziargyriou, N., Kamwa, I., Kundur, P., Martins, N., Paserba, J., Pourbeik, P., Sanchez-Gasca, J., Schulz, R., Stankovic, A., Taylor, C., & Vittal, V. (2005). Causes of the 2003 Major Grid Blackouts in North America and Europe, and Recommended Means to Improve System Dynamic Performance. *IEEE Transactions on Power Systems*, 20(4), 1922–1928.

Argonne National Laboratory (2007). MPICH2, an implementation of the Message-Passing Interface (MPI). `http://www-unix.mcs.anl.gov/mpi/`.

Armstrong, M., Martí, J. R., Linares, L. R., & Kundur, P. (2006). Multilevel MATE for efficient simultaneous solution of control systems and nonlinearities in the OVNI simulator. *Power Systems, IEEE Transactions on*, 21, 1250–1259. 3.

Arrillaga, J. & Watson, N. R. (2001). *Computer Modelling of Electrical Power Systems*. Wiley, 2 edition.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., & der Vorst, H. V. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM.

Beowulf.org (2009). Beowulf.org: Overview. http://www.beowulf.org/overview/index.html.

Blackford, L. S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., & Whaley, R. C. (2002). An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2), 135–151.

Brasch, F., Van Ness, J., & Kang, S.-C. (1979). The use of a multiprocessor network for the transient stability problem. In *Proc. PICA-79 Power Industry Computer Applications Conference IEEE* (pp. 337–344).

Bunch, J. R. & Rose, D. J. (1972). *Partitioning, Tearing, and Modification of Sparse Linear Systems*. Technical Report TR 72 - 149, Cornell University.

Chai, J. & Bose, A. (1993). Bottlenecks in parallel algorithms for power system stability analysis. *IEEE Trans. Power Syst.*, 8(1), 9–15.

Chai, J., Zhu, N., Bose, A., & Tylavsky, D. (1991). Parallel newton type methods for power system stability analysis using local and shared memory multiprocessors. *IEEE Trans. Power Syst.*, 6(4), 1539–1545.

Crow, M. & Ilić, M. (1990). The parallel implementation of the waveform relaxation method for transient stability simulations. *IEEE Trans. Power Syst.*, 5(3), 922–932.

Culler, D. E., Liu, L. T., Martin, R. P., & Yoshikawa, C. (1996). LogP performance assessment of fast network interfaces. *IEEE Micro*, (pp. 35–43).

Davis, T. A. (2004). A column pre-ordering strategy for the unsymmetric-pattern multifrontalmethod. *ACM Trans. Math. Softw.*, 30(2), 165–195.

De Rybel, T., Tomim, M. A., Singh, A., & Martí, J. R. (2008). An introduction to open-source linear algebra tools and parallelisation for power system applications. In *Electrical Power & Energy Conference, Vancouver, Canada*.

Decker, I., Falcão, D., & Kaszkurewicz, E. (1992). Parallel implementation of a power system dynamic simulation methodology using the conjugate gradient method. *Power Systems, IEEE Transactions on*, 7(1), 458–465.

Decker, I., Falcão, D., & Kaszkurewicz, E. (1996). Conjugate gradient methods for power system dynamic simulation on parallel computers. *Power Systems, IEEE Transactions on*, 11(3), 1218–1227.

Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., & Liu, J. W. H. (1999). A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20, 720—755.

Dommel, H. & Sato, N. (1972). Fast transient stability soultions. *IEEE Transactions on Power Apparatus and Systems*, PAS-91, 1643–1650.

Dommel, H. W. (1996). *EMTP Theory Book*. Vancouver, British Columbia, Canada: Microtran Power System Analysis Corporation, 2nd edition.

Enns, M., Tinney, W., & Alvarado, F. (1990). Sparse matrix inverse factors. *Power Systems, IEEE Transactions on*, 5(2), 466–473.

Ernst, D., Ruiz-Vega, D., Pavella, M., Hirsch, P., & Sobajic, D. (2001). A unified approach to transient stability contingency filtering, ranking and assessment. *IEEE Trans. Power Syst.*, 16(3), 435–443.

Esmaeili, S. & Kouhsari, S. (2007). A distributed simulation based approach for detailed and decentralized power system transient stability analysis. *Electric Power Systems Research*, 77(5-6), 673 – 684.

Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley.

Goto, K. (2006). Optimized GotoBLAS Libraries. Retrieved in January, 2007.

Goto, K. & Van De Geijn, R. A. (2008a). Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34, 1–25.

Goto, K. & Van De Geijn, R. A. (2008b). High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw.*, 35, 1–14.

Grainger, J. J. & Stevenson, W. D. (1994). *Power system analysis*. McGraw-Hill, Inc.

Grama, A., Gupta, A., Kumar, V., & Karypis, G. (2003). *Introduction to Parallel Computing*.

Gropp, W., Lusk, E., & Skjellum, A. (1999). *Using MPI : Portable Parallel Programming with the Message-Passing Interface*. Cambridge, Mass.: MIT Press.

Gropp, W., Lusk, E., & Sterling, T. (2003). *Beowulf Cluster Computing with Linux, 2nd Edition*. The MIT Press, 2 edition.

Happ, H. H. (1970). Diakoptics and piecewise methods. *IEEE Trans. Power App. Syst.*, (7), 1373–1382.

Happ, H. H. (1973). *Gabriel Kron and Systems Theory*. Schenectady, N.Y.: Union College Press.

Happ, H. H. (1974). Diakoptics-the solution of system problems by tearing. *Proc. IEEE*, 62(7), 930–940.

Hatcher, W., Brasch, F.M., J., & Van Ness, J. (1977). A feasibility study for the solution of transient stability problems by multiprocessor structures. *IEEE Trans. Power App. Syst.*, 96(6), 1789–1797.

Ho, C.-W., Ruehli, A., & Brennan, P. (1975). The modified nodal approach to network analysis. *IEEE Trans. Circuits Syst.*, 22(6), 504–509.

Hollman, J. & Martí, J. (2003). Real time network simulation with pc-cluster. *IEEE Trans. Power Syst.*, 18(2), 563–569.

Hong, C. & Shen, C. (2000). Implementation of parallel algorithms for transient stability analysis on a message passing multicomputer. In *Power Engineering Society Winter Meeting, 2000. IEEE*, volume 2 (pp. 1410–1415 vol.2).

Huang, J. & Wing, O. (1979). Optimal parallel triangulation of a sparse matrix. *Circuits and Systems, IEEE Transactions on*, 26(9), 726–732.

IEEE (1993). IEEE standard for scalable coherent interface (SCI).

Ilić-Spong, M., Crow, M. L., & Pai, M. A. (1987). Transient stability simulation by waveform relaxation methods. *IEEE Trans. Power Syst.*, 2(4), 943–949.

Juarez T., C., Castellanos, R., Messina, A., & Gonzalez, A. (2007). A higher-order newton method approach to computing transient stability margins. In R. Castellanos (Ed.), *Proc. 39th North American Power Symposium NAPS '07* (pp. 360–367).

Karypis, G. & Kumar, V. (1998a). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20, 359–392.

Karypis, G. & Kumar, V. (1998b). *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*. University of Minnesota, Department of Computer Science / Army HPC Research Center. Retrieved in October, 2006.

Kassabalidis, I. N. (2002). Dynamic security border identification using enhanced particle swarm optimization. *Power Systems, IEEE Transactions on*, 17(3), 723–729.

Kielmann, T., Bal, H. E., & Verstoep, K. (2000). Fast measurement of LogP parameters for message passing platforms. *of Lecture Notes in Computer Science*, 1800, 1176–1183.

Krause, P. C., Wasynczuk, O., & Sudhoff, S. D. (2002). *Analysis of Electric Machinery and Drive Systems*. Piscataway, NJ; New York: IEEE Press; Wiley-Interscience.

Kron, G. (1953). A method to solve very large physical systems in easy stages. *Circuit Theory, IRE Transactions on*, 2(1), 71–90.

Kron, G. (1963). *Diakoptics: The Piecewise Solution of Large-Scale Systems*. (London): Macdonald & Co.

Kundur, P. (1994). *Power System Stability and Control*. New York: McGraw-Hill.

Kundur, P. & Dandeno, P. (1983). Implementation of advanced generator models into power system stability programs. *power apparatus and systems, ieee transactions on*, PAS-102(7), 2047–2054.

Kundur, P., Morison, G., & Wang, L. (2000). Techniques for on-line transient stability assessment and control. In *Proc. IEEE Power Engineering Society Winter Meeting*, volume 1 (pp. 46–51).

La Scala, M., Bose, A., Tylavsky, D., & Chai, J. (1990a). A highly parallel method for transient stability analysis. *Power Systems, IEEE Transactions on*, 5(4), 1439–1446.

La Scala, M., Brucoli, M., Torelli, F., & Trovato, M. (1990b). A gauss-jacobi-block-newton method for parallel transient stability analysis. *IEEE Trans. Power Syst.*, 5(4), 1168–1177.

La Scala, M., Sblendorio, G., Bose, A., & Wu, J. (1996). Comparison of algorithms for transient stability simulations on shared and distributed memory multiprocessors. *IEEE Trans. Power Syst.*, 11(4), 2045–2050.

La Scala, M., Sblendorio, G., & Sbrizzai, R. (1994). Parallel-in-time implementation of transient stability simulations on a transputer network. *IEEE Trans. Power Syst.*, 9(2), 1117–1125.

La Scala, M., Sbrizzai, R., & Torelli, F. (1991). A pipelined-in-time parallel algorithm for transient stability analysis. *IEEE Trans. Power Syst.*, 6(2), 715–722.

Lau, K., Tylavsky, D., & Bose, A. (1991). Coarse grain scheduling in parallel triangular factorization and solution of power system matrices. *Power Systems, IEEE Transactions on*, 6(2), 708–714.

Li, X. S. & Demmel, J. W. (2002). SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *Lawrence Berkeley National Laboratory*. Paper LBNL-49388.

Li, X. S., Demmel, J. W., & Gilbert, J. R. (2003). *SuperLU Users' Guide*. The Regents of the University of California, through Lawrence Berkeley National Laboratory. Retrieved in October, 2006.

Mansour, Y., Vaahedi, E., Chang, A., Corns, B., Garrett, B., Demaree, K., Athay, T., & Cheung, K. (1995). BC Hydro's on-line transient stability assessment (TSA) model development, analysis and post-processing. *IEEE Trans. Power Syst.*, 10(1), 241–253.

Marceau, R. & Soumare, S. (1999). A unified approach for estimating transient and long-term stability transfer limits. *IEEE Trans. Power Syst.*, 14(2), 693–701.

Martí, J. R., Linares, L. R., Hollman, J. A., & Moreira, F. A. (2002). OVNI: Integrated software/hardware solution for real-time simulation of large power systems. In *Conference Proceedings of the 14th Power Systems Computation Conference, PSCC02, Sevilla, Spain.*

MPI Standard (2008). A Message-Passing Interface Standard. Retrieved [August 8, 2008] from `http://www-unix.mcs.anl.gov/mpi/`.

NICEVT (2005). Message-passing interface MPI2 over high speed net SCI. Retrieved [October 1, 2006] from `http://www.nicevt.ru/download/index.html?lang=en`.

Ogbuobiri, E., Tinney, W., & Walker, J. (1970). Sparsity-directed decomposition for gaussian elimination on matrices. *IEEE Trans. Power App. Syst.*, PAS-89(1), 141–150.

Owens, John, D., Luebke, David, Govindaraju, Naga, Harris, Mark, Kruger, Jens, Lefohn, Aaron, E., Purcell, & Timothy, J. (2007). A survey of General-Purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1), 113, 80.

Pai, M. & Dag, H. (1997). Iterative solver techniques in large scale power system computation. In *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, volume 4 (pp. 3861–3866 vol.4).

Rescigno, T. N., Baertschy, M., Isaacs, W. A., & McCurdy, C. W. (1999). Collisional breakup in a quantum system of three charged particles. *Science*, 286(5449), 2474–2479.

Saad, Y. & Vorst, H. A. V. D. (2000). Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123, 1—33.

Sato, N. & Tinney, W. (1963). Techniques for exploiting the sparsity or the network admittance matrix. *IEEE Trans. Power App. Syst.*, 82(69), 944–950.

Shu, J., Xue, W., & Zheng, W. (2005). A parallel transient stability simulation for power systems. *Power Systems, IEEE Transactions on*, 20(4), 1709–1717.

Stott, B. (1979). Power system dynamic response calculations. *Proc. IEEE*, 67(2), 219–241.

Tinney, W., Brandwajn, V., & Chan, S. (1985). Sparse vector methods. *IEEE Trans. Power App. Syst.*, PAS-104(2), 295–301.

Tinney, W. & Walker, J. (1967). Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, 55(11), 1801–1809.

Tomim, M., Martí, J., & Wang, L. (2009). Parallel solution of large power system networks using the Multi-Area Thévenin Equivalents (MATE) algorithm. *International Journal of Electrical Power & Energy Systems*, In Press, Corrected Proof. DOI: 10.1016/j.ijepes.2009.02.002.

Tomim, M. A., Martí, J. R., & Wang, L. (2008). Parallel computation of large power system network solutions using the Multi-Area Thévenin Equivalents (MATE) algorithm. In *16th Power Systems Computation Conference, PSCC2008* Glasgow, Scotland.

Tylavsky, D. J., Bose, A., Alvarado, F., Betancourt, R., Clements, K., Heydt, G. T., Huang, G., Ilic, M., Scala, M. L., Pai, M., Pottle, C., Talukdar, S., Ness, J. V., & Wu, F. (1992). Parallel processing in power systems computation. *Power Systems, IEEE Transactions on*, 7(2), 629–638.

Vorst, H. A. V. D. & Chan, T. F. (1997). Linear system solvers: Sparse iterative methods. *Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering*, 4, 167—202.

Wang, F. (1998). Parallel-in-time relaxed newton method for transient stability analysis. *Generation, Transmission and Distribution, IEE Proceedings-*, 145(2), 155–159.

Wang, L. & Morison, K. (2006). Implementation of online security assessment. *Power and Energy Magazine, IEEE*, 4(5), 46–59.

Wing, O. & Huang, J. (1980). A computation model of parallel solution of linear equations. *Computers, IEEE Transactions on*, C-29(7), 632–638.

Wu, F. (1976). Solution of large-scale networks by tearing. *IEEE Trans. Circuits Syst.*, 23(12), 706–713.

Wu, J. Q. & Bose, A. (1995). Parallel solution of large sparse matrix equations and parallel power flow. *Power Systems, IEEE Transactions on*, 10(3), 1343–1349.

Wu, J. Q., Bose, A., Huang, J. A., Valette, A., & Lafrance, F. (1995). Parallel implementation of power system transient stability analysis. *IEEE Trans. Power Syst.*, 10(3), 1226–1233.

Xue, Y., Rousseaux, P., Gao, Z., Belhomme, R., Euxible, E., & Heilbronn, B. (1993). Dynamic extended equal area criterion. In P. Rousseaux (Ed.), *Proc. Joint International Power Conference Athens Power Tech APT 93*, volume 2 (pp. 889–895).

Appendix A

LU Factorization

The LU factorization is mathematically equivalent to the Gaussian elimination and allows to express a permuted version of a specific complex system matrix $\mathbf{A}$ in terms of its lower and upper-diagonal factors, namely, $\mathbf{L}$ and $\mathbf{U}$. Once the factors $\mathbf{L}$ and $\mathbf{U}$ are formed, two new triangular linear systems are generated, which in turn, can be solved by forward and backward substitutions.

A.1 Problem Formulation

Let (A.1) represent a generic complex linear system, where $\mathbf{A}$ is a complex $n \times n$ matrix, and, $\mathbf{B}$ and $\mathbf{X}$ are, also complex, $n \times m$ matrices, where the last one is unknown.

$$\mathbf{A}\,\mathbf{X} = \mathbf{B} \tag{A.1}$$

In order to solve this system for $\mathbf{X}$, one alternative is to factorize $\mathbf{A}$ into two triangular matrices, namely $\mathbf{L}$ and $\mathbf{U}$, as stated in (A.2).

$$\mathbf{P}^r\,\mathbf{A}\,\mathbf{P}^c = \mathbf{L}\,\mathbf{U} \tag{A.2}$$

where $\mathbf{P}^r$ and $\mathbf{P}^c$ are a row and column permutation matrices, respectively. In the case $\mathbf{A}$ is dense, only $\mathbf{P}^r$ may be employed in order to perform diagonal partial pivoting, while both $\mathbf{P}^r$ and $\mathbf{P}^c$ are needed in case total pivoting is required. In case the matrix $\mathbf{A}$ large and sparse, $\mathbf{P}^r$ still performs partial pivoting during the factorization, whereas $\mathbf{P}^c$ is selected in such manner that it minimizes the number of added non-zeros (or simply *fill-ins*) in the structure of $\mathbf{P}^r\,\mathbf{A}\,\mathbf{P}^c$, and consequently, in the structure of $\mathbf{L} + \mathbf{U}$.

Pre-multiplying (A.1) by $\mathbf{P}^r$ and making $\mathbf{X} = \mathbf{P}^c\mathbf{Z}$ leads to the following expression

$$(\mathbf{P}^r\,\mathbf{A}\,\mathbf{P}^c)\,\mathbf{Z} = \mathbf{P}^r\,\mathbf{B} \tag{A.3}$$

where the product inside the brackets can be substituted by the product $\mathbf{LU}$, according to

(A.2), which in turn yields

$$\mathbf{L}\,\mathbf{U}\,\mathbf{Z} = \mathbf{P}^r\,\mathbf{B} \tag{A.4}$$

The new linear system obtained above can then be split into two interdependent triangular linear systems, defined by $\mathbf{L}$ and $\mathbf{U}$, whose result is the matrix $\mathbf{Z}$. Lastly, the solution of the original linear system, $\mathbf{X}$, can be obtained by permuting the rows of $\mathbf{Z}$ according to $\mathbf{P}^c$. Thus, introducing a new $n \times m$ matrix $\mathbf{W}$, which will keep the solution of the lower diagonal system defined by $\mathbf{L}$, this procedure is summarized as follows.

$$\mathbf{P}^r\,\mathbf{A}\,\mathbf{P}^c = \mathbf{L}\,\mathbf{U} \qquad \textit{(LU factorization)} \tag{A.5a}$$

$$\mathbf{L}\,\mathbf{W} = \mathbf{P}^r\,\mathbf{B} \qquad \textit{(forward substitution)} \tag{A.5b}$$

$$\mathbf{U}\,\mathbf{Z} = \mathbf{W} \qquad \textit{(backward substitution)} \tag{A.5c}$$

$$\mathbf{X} = \mathbf{P}^c\,\mathbf{Z} \qquad \textit{(row permutation)} \tag{A.5d}$$

A.2 LU Factorization Process

Assuming that the complex matrix $\mathbf{A}$ is already ordered, its $\mathbf{L}$ and $\mathbf{U}$ factors can be obtained in the manner very similar to the one synthetized in (A.6). Firstly, one separates the first row and column of the matrix $\mathbf{A}$, as shown in (A.6a). As a consequence, the previous partitioned matrix can be further rewritten, as in (A.6b), in the form of a multiplication of two other matrices matrices, namely $\mathbf{L}_1\mathbf{U}_1$. Note that $\mathbf{L}_1$ is already lower diagonal, while $\mathbf{U}_1$ is the matrix $\mathbf{A}$ with its first column eliminated by Gaussian elimination.

$$\mathbf{A} = \left[\begin{array}{c|c} a_1 & (\mathbf{r}_1)^T \\ \hline \mathbf{c}_1 & \mathbf{A}_1 \end{array}\right] = \tag{A.6a}$$

$$= \left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline & 1 & & \\ \frac{1}{a_1}\mathbf{c}_1 & & \ddots & \\ & & & 1 \end{array}\right] \left[\begin{array}{c|c} a_1 & (\mathbf{r}_1)^T \\ \hline \mathbf{0} & \mathbf{A}_1 - \frac{1}{a_1}\mathbf{c}_1\,(\mathbf{r}_1)^T \end{array}\right] = \tag{A.6b}$$

$$= \mathbf{L}_1\,\mathbf{U}_1 \tag{A.6c}$$

One can then repeat this procedure to matrices $\mathbf{A}_k = \mathbf{A}_{k-1} - \frac{1}{a_k}\mathbf{c}_k\,(\mathbf{r}_k)^T$ with $k = 2, \ldots, n-1$, which are in fact formed by means of recursive rank-1 updates. Such procedure

ultimately generates the following result.

$$\mathbf{A} = \mathbf{L}_1 \mathbf{L}_2 \ldots \mathbf{L}_{n-1} \mathbf{U}_{n-1} = \mathbf{L}\,\mathbf{U} \tag{A.7}$$

where each $\mathbf{L}_k$ is as follows.

$$\mathbf{L}_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \frac{1}{a_k}\mathbf{c}_k & \ddots & \\ & & & & 1 \end{bmatrix} \tag{A.8}$$

As a final result, the $\mathbf{U}_{n-1}$ is in the upper-diagonal triangular matrix $\mathbf{U}$, whereas the lower-diagonal triangular $\mathbf{L}$ equals to the product of all $\mathbf{L}_k$ with $k = 1, \ldots, n-1$, i.e., $\mathbf{L} = \prod_{k=1}^{n-1} \mathbf{L}_k$.

A.3 Computational Complexity of LU Factorization and Solution

As presented by (Bunch & Rose, 1972), assume the sparse matrix $\mathbf{M} = \mathbf{P}^r\mathbf{A}\mathbf{P}^c$ that also includes all fill-ins introduced in the structure $\mathbf{P}^r\mathbf{A}\mathbf{P}^c$, according to (A.2). This matrix $\mathbf{M}$ is known as *perfect elimination matrix*, since no fill-ins are introduced when decomposed into its $\mathbf{L}$ and $\mathbf{U}$ factors. In this context, Table A.1 can be generated, based on some metrics defined in (Alvarado, 1976) for the perfect elimination matrix $\mathbf{M}$. These metrics definitions are repeated below for convinience.

$$\tau_u = \sum_{i=1}^{n-1} r_i \tag{A.9}$$

$$\alpha = \sum_{i=1}^{n-1} r_i c_i \tag{A.10}$$

In the definitions (A.9) and (A.10), r_i is the number of non-zero elements in the i^{th} row of the matrix $\mathbf{U}$ above its main diagonal and c_i is the number of non-zero elements in the i^{th} column of the matrix $\mathbf{L}$ below its main diagonal. One can readily verify that τ_u equals the total number of off-diagonal non-zeros of $\mathbf{U}$. The metric α is the sum of all multiplications needed to perform all rank-1 updates on the LU structure, as shown in the previous section.

On a further note, for symmetric and topologically symmetric matrices, and consequently dense matrices, r_i is the same as c_i. The operation count for dense systems is also included in Table A.1, which were obtained by making $r_i = c_i = n - i$.

Table A.1. Operation count for sparse and dense LU factorization and solution for unsymmetric, but topologicaly symmetric, systems.

Task	Sparse			Dense		
	add	**mult**	**div**	**add**	**mult**	**div**
LU fact.	α	α	τ_u	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$	$\frac{n^2}{2} - \frac{n}{2}$
LU sol.	$2\tau_u$	$2\tau_u$	n	$n^2 - n$	$n^2 - n$	n

Although the operation counts summarized on Table A.1 provides an exact means to determine the required computational effort for both sparse LU factorizations and solutions, they completely rely on the sparsity pattern of a specific prefect elimination matrix $\mathbf{M}$. Therefore, a general way to, at least, predict the required computational effort of sparse operations is desirable. Bearing this need in mind, in the following subsection, one way to predict the number of operations will be studied.

A.3.1 Expected Computational Complexity of Sparse LU Factorization

Suppose that the $n \times n$ matrix $\mathbf{M} = \{m_{ij}\}$ is a perfect elimination matrix associated with the generic matrix $\mathbf{A}$, which is assumed to be unsymmetric but topology-symmetric, and has no zeros in its main diagonal. Moreover, each upper triangular location is also assumed to have an equal probability of being non-zero, as given by (A.11).

$$p = p(m_{ij} \neq 0) = \frac{2\tau_u}{n^2 - n} \tag{A.11}$$

The fact that there are $n - i$ possible upper diagonal elements in row i of $\mathbf{M}$ with equal probability of being non-zero, given by (A.11), indicates that the number of non-zero elements r_i, in row i above the diagonal of $\mathbf{M}$ is a binomially distributed random variable. Therefore, according to basic statistics manipulation, the expected value and variance of r_i

are given as follows.

$$E(r_i) = (n-i)p \tag{A.12}$$

$$\sigma^2(r_i) = (n-i)(1-p)p \tag{A.13}$$

Now, recalling that the operation counts given in Table A.1 depend on the metrics τ_u and α, defined in (A.9) and (A.10), respectively, the expected value for these metrics can also be obtained as follows.

$$E(\tau_u) = \sum_{i=1}^{n-1} E(r_i) = \tau_u \tag{A.14}$$

$$\begin{aligned} E(\alpha) &= \sum_{i=1}^{n-1} E\left({r_i}^2\right) = \sum_{i=1}^{n} \left[\sigma^2(r_i) + E^2(r_i)\right] = \\ &= \frac{4}{3}\frac{(n-2)}{(n-1)\,n}{\tau_u}^2 + \tau_u \end{aligned} \tag{A.15}$$

Defining the new parameter ρ, given in (A.16), which represents the ratio of branches to buses[11], one can rewrite (A.14) and (A.15) as shown in (A.17) and (A.18), respectively.

$$\rho = \frac{\text{number of off-diagonal nonzeros}}{\text{number of nodes}} = \frac{\tau_u}{n} \tag{A.16}$$

which gives

$$E(\tau_u) = n\rho \tag{A.17}$$

$$E(\alpha) = \frac{4}{3}\frac{(n-2)}{(n-1)}n\rho^2 + n\rho \tag{A.18}$$

The choice of expressing the above expected values in terms of branch-bus ratio ρ instead of τ_u is due to the fact that buses in electric networks are usually connected to only a few other neighboring buses. Thus, expressing τ_u in terms of ρ provides a measure of how interconnected the system under study is.

Combining Table A.1 and equations (A.17) and (A.18), expressions for predicting the floating-point operations required to perform sparse LU factorizations, backward and forward substitutions can be deduced, which are shown in Table A.2. Since only complex systems have been assumed so far, the number of floating-point operations needed for each basic complex operation, i.e., addition, subtraction, multiplication and divisions, used for

[11]Note that the branch-bus ratio ρ is associate with the perfect elimination matrix $\mathbf{M}$ and not the system matrix $\mathbf{A}$. Therefore, the number of branches is always dependent on the ordering scheme adopted to reduce fill-ins in the original matrix $\mathbf{A}$ structure

obtaining Table A.2 is provided in Table A.3.

Notice that the extra column in Table A.2, related to large power systems, takes into consideration that the term $\frac{(n-2)}{(n-1)} \approx 1$ when $n \gg 2$, which leads to a simpler expression to represent $E(\alpha)$.

Table A.2. Floating-point operation count for sparse and dense LU factorization and solution for unsymmetric, but topologicaly symmetric, systems.

Task	**Sparse**	**Sparse** $(n \gg 2)$	**Dense**
LU factorization	$\left(\frac{32}{3}\frac{(n-2)}{(n-1)}\rho^2 + 19\rho\right) n$	$\left(\frac{32}{3}\rho^2 + 19\rho\right) n$	$\frac{8}{3}n^3 + \frac{3}{2}n^2 - \frac{25}{6}n$
LU solution	$(16\rho + 11)\, n$	$(16\rho + 11)\, n$	$8n^2 + 3n$

Table A.3. Complex basic operations requirements in terms of floating-point count.

Complex Operation	**Mathematical Representation**	**Flop Count**
addition/subtraction	$(a + jb) \pm (c + j\,d) = (a + c) \pm j\,(b + d)$	2 add/subtr. **2 flops**
multiplication	$(a + j\,b) \times (c + j\,d) = (ac - bd) + j\,(bc + ad)$	2 add. 4 mult. **6 flops**
division	$\frac{a + j\,b}{c + j\,d} = \frac{(ac + bd) + j\,(bc - ad)}{c^2 + d^2}$	3 add. 6 mult. 2 div. **11 flops**

Appendix B

Transient Stability Solution Techniques

As extensively described in the literature (Dommel & Sato, 1972; Stott, 1979; Kundur, 1994), the power system transient stability model can be summarized by the set of non-linear differential-algebraic equations shown in (B.1).

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{v}) \tag{B.1a}$$

$$\mathbf{Y}\,\mathbf{v} = \mathbf{i}(\mathbf{x}, \mathbf{v}) \tag{B.1b}$$

where, $\mathbf{x}$ represents a vector with dynamic variables (or, state variables), whose first derivatives $\dot{\mathbf{x}}$ are normally dependent on $\mathbf{x}$ themselves and the vector with nodal voltages $\mathbf{v}$. In addition, $\mathbf{i}$ represents a vector function that defines the nodal current injections, which also depend on the variable states $\mathbf{x}$ and the nodal voltages $\mathbf{v}$. Lastly, $\mathbf{Y}$ represents the complex-valued nodal admittance matrix of the system under study.

According to the literature (Dommel & Sato, 1972; Stott, 1979; Kundur, 1994), the many possible alternatives for solving (B.1) simultaneously in time are categorized in terms of

(a) the way in which (B.1a) and (B.1b) are interfaced;

(b) the integration method, which can be explicit or implicit;

(c) the technique used for solving the algebraic equations, that may be linear or non-linear. For cases when (B.1b) is linear, sparsity-based direct solutions should be employed, whereas for non-linear cases, either the Gauss-Seidel or the Newton-Raphson methods should be used.

B.1 Problem Discretization

In this method, (B.1a) is firstly discretized according to some integration rule. Due to its inherent numerical stability, the trapezoidal rule is chosen (Dommel, 1996; Dommel & Sato,

1972). This procedure is shown in (B.2).

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(t-\Delta t) + \int_{t-\Delta t}^{t} \mathbf{f}\big(\mathbf{x}(\xi),\mathbf{v}(\xi)\big)d\xi \approx \\ &\approx \mathbf{x}(t-\Delta t) + \frac{\Delta t}{2}\Big[\mathbf{f}\big(\mathbf{x}(t),\mathbf{v}(t)\big) + \mathbf{f}\big(\mathbf{x}(t-\Delta t),\mathbf{v}(t-\Delta t)\big)\Big]\end{aligned} \tag{B.2}$$

Collecting the present and past values yields (B.3), where $\mathbf{x}_h(t)$ represents the *history term* of the state vector $\mathbf{x}$, which is known at time t.

$$\mathbf{x}(t) = \frac{\Delta t}{2}\,\mathbf{f}\big(\mathbf{x}(t),\mathbf{v}(t)\big) + \mathbf{x}_h(t) \tag{B.3a}$$

$$\mathbf{x}_h(t) = \mathbf{x}(t-\Delta t) + \frac{\Delta t}{2}\,\mathbf{f}\big(\mathbf{x}(t-\Delta t),\mathbf{v}(t-\Delta t)\big) \tag{B.3b}$$

There are many solution variations described in the literature that combine different integration methods and algebraic equations solutions. A number of these methods are summarized in (Stott, 1979). However, all variations still fall into two major categories: *alternating* (or *partitioned*) and *simultaneous* solution approaches. Since the alternating solution approach is described and discussed in Section 4.1, only the simultaneous solution will be discussed next.

B.2 Simultaneous Solution Approach

One of the most prominent methods pertaining to this class is undoubtedly the Newton-Raphson method. The method is often formulated by means of residual vector functions $\mathbf{g}\big(\mathbf{x}(t),\mathbf{v}(t)\big)$ and $\mathbf{h}\big(\mathbf{x}(t),\mathbf{v}(t)\big)$, given in (B.4), which are obtained from combining the discretized differential equations (B.3) and the network algebraic equations (B.1b).

$$\mathbf{g}\big(\mathbf{x}(t),\mathbf{v}(t)\big) = \mathbf{x}(t) - \frac{\Delta t}{2}\,\mathbf{f}\big(\mathbf{x}(t),\mathbf{v}(t)\big) - \mathbf{x}_h(t) = \mathbf{0} \tag{B.4a}$$

$$\mathbf{h}\big(\mathbf{x}(t),\mathbf{v}(t)\big) = \mathbf{Y}\,\mathbf{v}(t) - \mathbf{i}\big(\mathbf{x}(t),\mathbf{v}(t)\big) = \mathbf{0} \tag{B.4b}$$

In the Newton-Raphson method, the iterates for $\mathbf{x}(t)$ and $\mathbf{v}(t)$ can be obtained using (B.5).

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \boldsymbol{\Delta}\mathbf{x}^{k} \tag{B.5a}$$

$$\mathbf{v}^{k+1} = \mathbf{v}^{k} + \boldsymbol{\Delta}\mathbf{v}^{k} \tag{B.5b}$$

where the mismatches $\boldsymbol{\Delta}\mathbf{x}^k$ and $\boldsymbol{\Delta}\mathbf{v}^k$ are computed from the linear system (B.6)[12].

$$\begin{bmatrix} \mathbf{A_d} & \mathbf{B_d} \\ \mathbf{C_d} & \mathbf{Y} + \mathbf{Y_d} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta}\mathbf{x}^k \\ \boldsymbol{\Delta}\mathbf{v}^k \end{bmatrix} = - \begin{bmatrix} \mathbf{g}\big(\mathbf{x}^k, \mathbf{v}^k\big) \\ \mathbf{h}\big(\mathbf{x}^k, \mathbf{v}^k\big) \end{bmatrix} \tag{B.6}$$

and

$$\mathbf{A_d} = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_k = \mathbf{U} - \frac{\Delta t}{2}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\big(\mathbf{x}^k, \mathbf{v}^k\big) - \mathbf{x}_h \qquad \mathbf{B_d} = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{v}}\right|_k = -\frac{\Delta t}{2}\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\big(\mathbf{x}^k, \mathbf{v}^k\big)$$
$$\mathbf{C_d} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_k = -\frac{\partial \mathbf{i}}{\partial \mathbf{x}}\big(\mathbf{x}^k, \mathbf{v}^k\big) \qquad \mathbf{Y_d} = -\frac{\partial \mathbf{i}}{\partial \mathbf{x}}\big(\mathbf{x}^k, \mathbf{v}^k\big)$$

Based on the fact that dynamic devices are interconnected through the network and not directly to each other, state variables associated with a device do not depend on other's state variables. Therefore, $\mathbf{A_d}$ is a block-diagonal matrix, denoted as follows for a power system with m dynamic devices.

$$\mathbf{A_d} = \begin{bmatrix} \mathbf{A}_{d1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{d2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{A}_{dm} \end{bmatrix}$$

Applying Gaussian elimination to the Jacobian matrix in (B.6) yields (B.7a) and (B.7b), which enable the solution of $\boldsymbol{\Delta}\mathbf{x}^k$ and $\boldsymbol{\Delta}\mathbf{v}^k$, respectively.

$$\boldsymbol{\Delta}\mathbf{x}^k = -\mathbf{A_d}^{-1}\big[\mathbf{g}\big(\mathbf{x}^k, \mathbf{v}^k\big) + \mathbf{B_d}\,\boldsymbol{\Delta}\mathbf{v}^k\big] \tag{B.7a}$$
$$\big(\mathbf{Y} + \mathbf{Y_d} - \mathbf{C_d}\mathbf{A_d}^{-1}\mathbf{B_d}\big)\,\boldsymbol{\Delta}\mathbf{v}^k = -\mathbf{h}\big(\mathbf{x}^k, \mathbf{v}^k\big) + \mathbf{C_d}\mathbf{A_d}^{-1}\mathbf{g}\big(\mathbf{x}^k, \mathbf{v}^k\big) \tag{B.7b}$$

Once $\boldsymbol{\Delta}\mathbf{x}^k$ and $\boldsymbol{\Delta}\mathbf{v}^k$ are calculated, they can be used to find new iterates $\mathbf{x}^{k+1}$ and $\mathbf{v}^{k+1}$ according to (B.5). In the sequence, $\mathbf{g}\big(\mathbf{x}^{k+1}, \mathbf{v}^{k+1}\big)$ and $\mathbf{h}\big(\mathbf{x}^{k+1}, \mathbf{v}^{k+1}\big)$ can be computed. Analogously to the partitioned method, the iterative process continues until the difference between two successive solutions of $\mathbf{x}(t)$ and $\mathbf{v}(t)$ is within a certain tolerance.

Although the full Newton-Raphson method is a powerful method for solving the transient stability non-linear equations, it is very computationally expensive, due to its dependency on the time-varying reduced Jacobian $\mathbf{J}_R = \mathbf{Y} + \mathbf{Y_d} - \mathbf{C_d}\mathbf{A_d}^{-1}\mathbf{B_d}$. Such dependency requires a sparse matrix inversion every single step, which slows down the overall solution by at least an order of magnitude. In order to overcome such a drawback, in industrial-grade transient

[12]The matrix $\mathbf{U}$ represents an identity matrix.

stability programs, the Jacobian matrix $\mathbf{J}_R$ is assembled and factorized into its LU factors only at the beginning of a simulation and whenever topology changes occur in the systems or convergence problems are faced. Under such conditions, the algorithm is usually referred to as *Very DisHonest Newton* (VDHN) method.

B.3 Simultaneous versus Alternating Solution Approach

The just discussed simultaneous solution approach can be compared with the alternating approach in various aspects, such as modeling flexibility, computational requirements and convergence characteristics. These are discussed below:

- **Computational requirements:** Since the simultaneous approach is usually related to Newton-Raphson methods for solution of non-linear system of equations, it demands the dynamic models to be expressed in terms of real and imaginary parts. As a direct consequence, the problem, which was originally complex-value of order n, becomes a real problem of order $2n$. Moreover, this newly formed real system of equations has also the four times the number of non-zeros of its complex-valued counterpart. For instance, during the factorization process (Appendix A), the amount of floating-point operations can be expected to, at least, quadruple. Other disadvantages of the real formulation lie on the increased number of memory references and memory consumption.

- **Modeling flexibility:** In terms of modeling, the real form of the simultaneous solution yields greater flexibility over the complex form, inherent to the alternating solution. In the complex form, complex derivatives, required by the linearization process, often depend on complex conjugates of the same variable, which demands the inclusion of redundant equations to the problem. This disadvantage can be exemplified by a common constant power load. In this case, the current $\bar{I}_L$ injected by the load $P_0 - j\,Q_0$ into the system relates to the its voltage $\bar{V}_L$ according to (B.8). In order to include this model in a Newton-like solution procedure, one needs to find a relationship between variations of current $\Delta\bar{I}_L$ and voltage $\Delta\bar{V}_L$, which is not possible from (B.8) alone. Since, only (B.9) can be generated, the solution needs to add the equation associated with $\Delta\bar{V}_L^*$.

$$\bar{I}_L = -\frac{P_0 - j\,Q_0}{\bar{V}_L^*} \tag{B.8}$$

$$\Delta\bar{I}_L = \frac{P_0 - j\,Q_0}{\left(\bar{V}_L^*\right)^2}\,\Delta\bar{V}_L^* \tag{B.9}$$

- **Convergence characteristics:** Newton-like methods are known for their numerical robustness and quadratic convergence. However, these aspects are entirely true only when the Jacobian of the non-linear problem, expressed by the submatrices $\mathbf{A_d}$, $\mathbf{B_d}$, $\mathbf{C_d}$ and $\mathbf{Y_d}$ in (B.7), is updated at every iteration. In this case, factorization of the problem is also required at every iteration, which, for very large systems, may degrade considerably the overall performance of the algorithm. Dishonest variants, like the VDHN method, which keep the Jacobian constant unless topological changes or convergence difficulties occur, are often employed, as a tradeoff between computational burden and convergence strength of full Newton methods. Therefore, for very large systems, the alternating solution approach may become comparable with the simultaneous solution approach in terms of convergence.

Appendix C

Network Microbenchmarks

In order to optimize the performance of parallel applications, the underlying algorithms need to be translated in terms of simpler tasks, for which the performance on a given computing system is known. In this manner, bottlenecks can be identified and available resources properly allocated aiming at the improvent of the overall application.

Regarding communications in parallel computing systems, a few modeling approaches have been proposed in the literature. The LogP (Culler et al., 1996), PLogP (Kielmann et al., 2000) and LogGP (Alexandrov et al., 1995) are examples of communication models in distributed computing systems. The basic foundations of such models lie on the knowledge of certain network parameters, such as, latency L, overheads for sending, o_s, and receiving, o_r, and inter-message gaps g.

In this context, network microbenchmarks provide the means of systematically acquire the previous network parameters. In turn, microbenchmarks play an paramount role in assessing the performance of the hardware and software that makes the bridge between applications and network interfaces (Culler et al., 1996).

C.1 Procedure

As the starting point towards the microbenchmark suggested by Kielmann et al. (2000), the gap between zero-byte messages, $g(0)$, is first measured, following the Algorithm 1, given below. In this algorithm, each message consists of a send and receive pair.

According to Culler et al. (1996), three stages can be observed during this procedure. For a small number of messages M, the issued sending requests receive no replies. Hence, the cost of each of these first requests equals to the sending overhead o_s. As the number of messages M increase, the sender starts receiving replies and the message cost increases, since the receiver spends o_r for each reply. During this stage, the replies are separated by the time for successive messages to pass through the bandwidth bottleneck, i.e., by g. As M further increases, the number of messages eventually causes the network capacity limit to be reached. Under such a condition, any sending request will stall, until a reply is drained from the network. During this stage, the cost of each message is determined by the gap g.

Algorithm 1 Inter-message gap and sending overhead timing

```
t_start ← start timer
for k = 1 to M do
  if Sender then
    Send message
  else
    Receive message
  end if
end for
t_stop ← stop timer
t ← t_stop − t_start
t_issue ← t/M
```

The results for the measurement of $g(0)$ for the Gigabit Ethernet network available in the UBC's Power Systems Engineering Laboratory is shown in Figure C.1.

Next, the round trip time (RTT) for a given message is measured, following the Algorithm 2. Notice that both request message m_1 and reply message m_2 also consist of paired send and receive operations. Since, in the PLogP model, the $RTT(0)$ is equivalent to $2\big(L + g(0)\big)$, one only needs to calculate the latency L.

As suggested by Kielmann et al. (2000), the receiving overhead can be measured by the Algorithm 3. In this case, the receiving overhead o_r can be measured by adding a controlled waiting time Δ, such that receiving the m-byte message occurs immediately without further delay. Without this extra waiting time, an extra idle time would be added to the measurement. Therefore, making $\Delta > RTT(m)$ is, usually, conservative enough.

Algorithm 2 Round trip timing (RTT)

```
t_start ← start timer
for k = 1 to M do
  if Sender then
    Send message m_1
    Receive message m_2
  else
    Receive message m_1
    Send message m_2
  end if
end for
t_stop ← stop timer
t ← t_stop − t_start
RTT ← t/M
```

Algorithm 3 Receiving overhead o_r timings

```
o_r ← 0
for k = 1 to M do
  if Sender then
    Send zero-byte message
    Wait for time Δ > RTT(m)
    t_start ← start timer
    Receive m-byte message
    t_stop ← stop timer
    o_r ← o_r + t_stop − t_start
  else
    Receive zero-byte message
    Send m-byte message
  end if
end for
if Sender then
  o_r ← o_r/M
end if
```

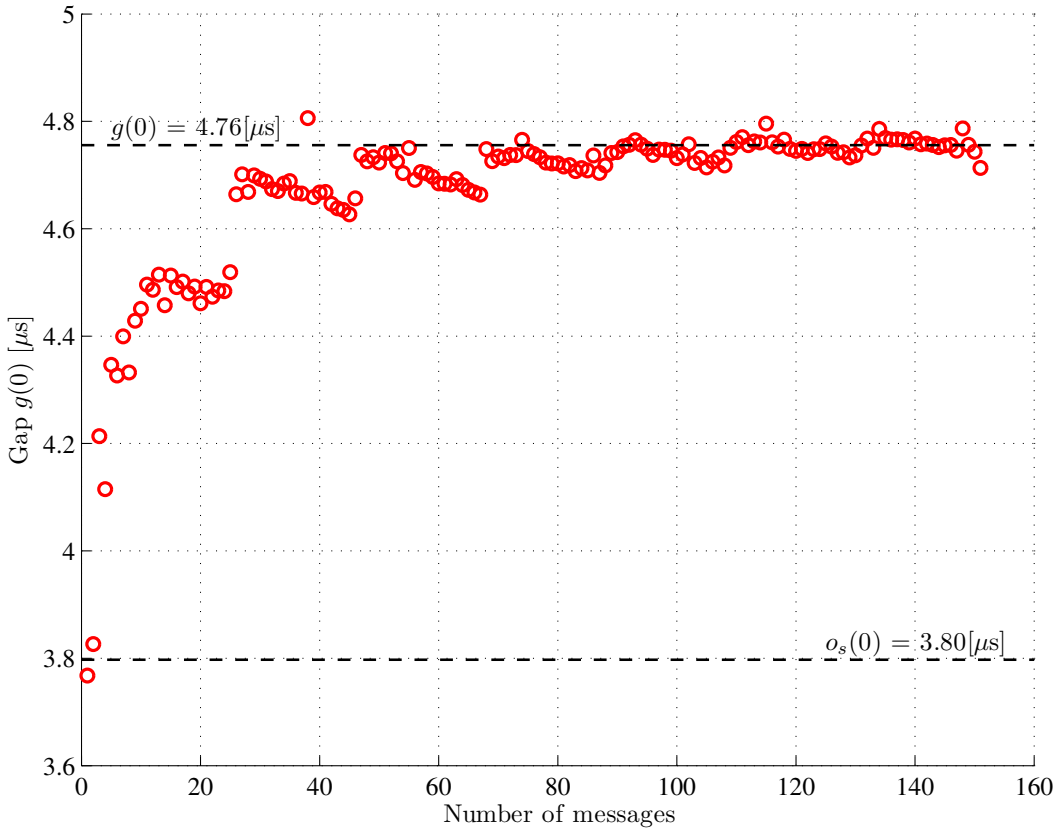


Figure C.1. Average issue time, yielded by Algorithm 1, for zero-byte messages communicated by MPI routines over a Gigabit Ethernet network.