Multi-Core Platform of Admittance Matrix Formation of Power Systems: Computational Time Assessment

[1]Gonzalez-Longatt, Francisco; [1]Montalvo, Martha Nohemi; [2]Andrade, Manuel A.; [2]Vazquez, Ernesto; [3]Chamorro, Harold R.; [4]Sood, Vijay K.

[1]Information Technology and Cybernetics - University of South-Eastern Norway, Norway
[2]School of Mechanical and Electrical Engineering - Universidad Autonoma de Nuevo Leon, Mexico
[3]KU Leuven, Katholieke Universiteit Leuven, Belgium
[4]University of Ontario, Canada

Gonzalez-Longatt, F., Acosta, M. N., Andrade, M., Vazquez, E., Chamorro, H. R., & Sood, V. K. (2020). Multi-Core Platform of Admittance Matrix Formation of Power Systems: Computational Time Assessment. In 2020 IEEE Electric Power and Energy Conference (EPEC), pp. 1-6. https://doi.org/10.1109/EPEC48502.2020.9320060

Publisher's version: DOI: 10.1109/EPEC48502.2020.9320060

Multi-Core Platform of Admittance Matrix Formation of Power Systems: Computational Time Assessment

Francisco Gonzalez-Longatt
Department of Electrical Engineering, InformationTechnology and Cybernetics
University of South-Eastern Norway
Porsgrunn, Norway
fglongatt@fglongatt.org

Martha N. Acosta
Department of Electrical Engineering, InformationTechnology and Cybernetics
University of South-Eastern Norway
Porsgrunn, Norway
Martha.Acosta@usn.no

Manuel Andrade
School of Mechanical and Electrical Engineering
Universidad Autonoma de Nuevo Leon
Nuevo Leon, Mexico
manuel.andradest@uanl.edu.mx

Ernesto Vazquez
School of Mechanical and Electrical Engineering
Universidad Autonoma de Nuevo Leon
Nuevo Leon, Mexico
evazquezmtz@gmail.com

Harold R. Chamorro
KU Leuven
Katholieke Universiteit Leuven
Leuven, Belgium
hr.chamo@ieee.org

Vijay K. Sood
Department of Electrical and Computer Engineering
University of Ontario
Ontario, Canada
vijay.sood@ontariotechu.ca

2020 IEEE Electric Power and Energy Conference (EPEC) | 978-1-7281-6489-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/EPEC48502.2020.9320060

***Abstract*—This paper presents a comparison of computational time required to build the admittance matrix of five test systems (ranging from 200 to 70,000 nodes) considering two formation approaches: element-by-element and matrix approach. The algorithms have been implemented in MATLAB™ and tested in four multi-core platforms. Implementations include sparse and dense matrix representation and parallel/non-parallel computing. Results show the matrix approach considering sparse representation and parallel computing is the best approach in computing time.**

Keywords—admittance matrix, computing time, hardware, power systems analysis, software.

I. Introduction

The analysis of even modest-sized power systems requires the use of appropriate tools to perform the calculation; it is where appropriate computer programs and well-designed software play a significant role [1]. Computational time spent in the power system study becomes a crucial element when the results are required in a very short time frame, especially if the application is desired to provide a response in real-time [2]. The availability of powerful computational hardware together with novel mathematical approaches used for network solution are two fundamental approaches in the development of real-time applications [3], [4].

The power system studies require the calculation of different types of matrices. The positive sequence admittance matrix is typically used in the calculation of the classical power flow calculations; also, the Jacobian matrix is needed when the load flow calculations are performed with the Newton-Raphson algorithm. The positive, negative, and zero sequence impedance matrices are calculated when the asymmetrical short circuit current calculations are required. However, many other matrices are used in power systems analysis; as a consequence, matrices formation and matrix algebra are two essential tools taken into consideration when developing real-time applications. The power flow study is the most frequently used in power system analysis because it is an essential tool for system planning and operation [5], [6]. It is used to determine the voltage, current, active, and reactive power and power factor in a power system; however, the classical load flow is used as input to much other power system analysis, for instance, as the input of the power system stability analysis, providing the steady-state conditions prior to a disturbance [1], [7], [8].

The power flow analysis finds its roots in the use of the *mesh-current* and *node-voltage* analysis methods described in introductory electrical circuit theory texts. However, the terms *loop* and *bus* are frequently used in place of mesh and node. Historically, the first computer attempts to solve the power flow problem was based in loop equations which did not exploit the capability of the computers. As a consequence, limited success was basically achieved due to the computational burden and limited computer memory available [9], [10], [11]. The node-voltage method or nodal analysis directly references bus quantities using the bus admittance matrix has been the most popularly used method by load flow analysis software. The nodal admittance matrix (or just admittance matrix) $\mathbf{Y}_{\mathbf{bus}}$ is a $N_{bus} \times N_{bus}$ matrix describing the nodal admittances of a N_{bus} buses power system [9], [12]. The admittance matrix has several properties; one of them is that the matrix is heavily weighted in the main diagonal and highly sparse. This is a consequence of the typical topology of a power system which has a relative absence of certain problem interconnections [13], [14].

Many scientific papers have focused on dealing with the problems of power system analysis of large-scale power systems. Over the years, many advances have dedicated their efforts on improvements in the load flow calculations in extensive networks [15], [16]; also, in the use of sparse matrix algebra to solving the problem of extensive power systems. Most recent advances include exploiting the value of the computational hardware [16]: multi-core CPU (central processing unit) [16] and many-core GPU (graphics processing unit). Traditionally, the computational load related to solving a very large-scale power system has relied on the use of a mainframe computer or specially designed computation architecture located in a university or a research lab. However, the modern mainstream of computation architectures in current market technically fall into two categories: *the multi-core CPU* and the *many-core GPU.* Although some discussion related to both architectures is

presented in this paper, the main focus is the computational time assessment on a multi-core CPU.

This scientific paper presents a univariable assessment of the performance of the multi-core platform in the processing of the admittance matrix. The computational time is used as the primary indicator to compare two algorithms of the admittance matrix formulation. The primary motivation behind investigating the mechanism of speeding up the admittance matrix formulation is because some problems like contingency analysis might require re-building or modifying the admittance matrix. However, some methods are already in the literature to modify the already created admittance matrix to simulate contingency; the authors are exploring some potential alternatives on the benefit of reformulating the whole admittance matrix.

The rest of the paper is organised as follows; Section II introduces the main features of the hardware and software used in power system application with particular emphasis on the admittance matrix implementation in this paper. Section III briefly presents the two methodologies implemented for the formation of the admittance matrix. One methodology is based on using element by element formation, and the second method uses an approach based on matrix manipulations, it is called 'Optimised code', in this paper. Then Section IV is dedicated to profusely test the two algorithms of admittance-matrix formation in four different multi-core platforms. Section V concludes and presents new research directions found during the course of this research paper. The main contribution of this paper is comparing two admittance matrix formation approaches by considering the computational time used by the algorithms to create the admittance matrix in four multi-core platforms. The input data is pre-processed in an isolated fashion and the computational time is not included in this assessment, the data structure is the same for both algorithms, and it consists of the connectivity of the power system and the series admittance. The performed experiments show that the Optimised code results much faster than the element-by-element admittance matrix formation. Although the latest is the most straightforward implementation of the admittance matrix implementation (in terms of coding), the computational time results very high when compared with the optimised code, and the reason accessing element-by-element the memory requires more CPU time that moving the data as a whole memory location area.

II. COMPUTATIONAL SYSTEMS

This section defines the main features of the hardware and software in use (or potential use) in power system analysis calculations using a digital computer.

A. Hardware

Many computational resources are available for power system analysis. Excluding dedicated mainframes and user-specific computers, the computational resources available for power system analysis includes *central processing unit* (CPU), *graphics processing unit* (GPU), and *Field-programmable gate arrays* (FPGAs). Another lesser-known alternative is the *Application Specific Integrated Circuit* (ASIC); this is an integrated circuit (IC) chip customised for a particular use rather than a general-purpose use. As a consequence of the specificity of the design, the efficiency in the calculation is optimised. One specific example of an ASIC is the *tensor processing unit* (TPU). The TPU concept was introduced by Google in 2016; it is a custom-developed ASIC used for the specific application of accelerating *artificial intelligence* (AI) and machine learning (ML) workloads. Other manufacturers of ML specific ASIC include Nervana, Graphcore, Wave systems, etc.

A *Field Programmable Gate Array* (FPGA) is a reconfigurable integrated circuit. The developer can configure the FPGA to become any circuit he/she wants to (as long as it fits on the FPGA). Therefore, the FPGA is a less common alternative to write a power system analysis software for an instruction-based architecture, such as a CPU or GPU. Although an FPGA can offer an increase in calculation speed, the developer is exposed to a very different workflow. A classical programmer uses instruction-based hardware most programmers are used to, i.e. like CPUs and GPUs, but FPGAs are instead configured by specifying a hardware circuit. As a consequence, the engineering cost is typically much higher than for instruction-based architectures. Finally, even if the FPGA becomes slightly more energy-efficient than GPUs, the development of software for FPGAs is still a lot more complicated than for GPUs. Well-known manufacturers of FPGAs include Xilinx and Intel, but more manufacturers are offering options, e.g. Arduino MKR Vidor 4000. A GPU is a programmable chip intended initially for graphics rendering. The modern GPU commenced in 1995; it was marked by the introduction of the first 3D add-in cards. Later, the GPU use expanded by the adoption of the 32-bit operating systems and affordable personal computers. However, 2007 defined the beginning of the era of general-purpose GPUs. The big manufacturers of general-purpose GPUs, Nvidia and ATI (now acquired by AMD), have been packing their graphics cards with ever-more capabilities. By 2007, Nvidia released its CUDA® (Compute Unified Device Architecture) development environment, and by 2009 the called OpenCL (*Open Computing Language*) became widely supported.

The CPU is responsible for calculations in a computer, and for controlling or supervising other parts of the computer. The CPU is designed to perform logical and floating-point operations on data held in the computer memory. The CPU has been the most widely used instruction-based architecture, especially for power system applications. The speed of the processor clock defines how quickly the CPU functions. The CPU is designed to handle a wide range of tasks quickly but are limited in the concurrency of tasks that can be running. Manufacturers of CPU include Intel, AMD, ARM, CEVA. The GPU has a highly parallel structure that makes them more effective than general-purpose CPUs for algorithms where the processing of large blocks of data is done in parallel. However, high clock speed makes the CPU quite attractive when sequential programming is used. Although CPU and GPU share the concept of the core, they are not equivalent to each other; GPU cores perform specialised operations whereas CPU cores are designed for general-purpose programs. The design architecture of the CPU has only a few cores with lots of cache memory that can handle a few software threads at a time; for instance, an Inter® Core® i7 - 8850H has 8 cores and 16 threads. On the other hand, a modern GPU is composed of hundreds of cores that can handle thousands of threads simultaneously; for instance, RTX 2080Ti has a total of 4,352 CUDA Cores and 102 threads.

B. Software

The software complements the hardware in a computational system, and it is a collection of data or

computer instructions that tell the computational system how to work. Contrary to a mainframe computer, 70% of the desktop and laptop computers around the world use Microsoft® Windows®. As a consequence, this paper focused on computational systems using Windows® 10 as the operating system. However, as this paper is looking into the assessment of two different approaches of admittance matrix formation, the programming language, and the approach to use in the problem solution are crucial aspects to consider. In this paper, two approaches to programming are considered: (i) Parallel programming and (ii) Sequential programming. The first involves the concurrent computation or simultaneous execution of processes or threads at the same time. The latter involves a consecutively ordered execution of processes. As the sequential approach is widely explained in the literature, it is not explained here.

C. Parallel Computing

The use of parallel computing is especially attractive when solving computationally- and data-intensive problems and there is the availability of multi-core processors, GPUs, and computer clusters. Parallel computing is advantageous when the problem to solve requires many calculations simultaneously. However, the critical point here is that large problems must be able to be split into smaller problems, which are then solved at the same time. Parallel computing requires the combination of hardware and software able to perform the task in a parallel fashion. Regarding hardware, parallel computer architecture exists in a wide variety of parallel computers: (i) Multi-core computing (ii) Symmetric multiprocessing, (iii) Distributed computing and (iv) Massively parallel computing. In this paper, the interest is centred in the use of multi-core architecture. It allows executing program instructions in parallel. The cores are integrated onto multiple dies in a single chip package or onto a single integrated circuit die and may implement architectures such as multithreading, superscalar, vector, or very long instruction word (VLIW). The multi-core architectures are grouped depending on the characteristics the cores: homogeneous that includes only identical cores or heterogeneous that includes cores that are not identical. There is a wide range of enabling parallel computing software available. The use of some programming tools like the Parallel Computing Toolbox™ of MATLAB®, allows users to parallelise the problem-solution process without the use of more complex elements like CUDA (*Compute Unified Device Architecture*) or MPI (*Message Passing Interface*) programming. NVIDIA® originally introduced CUDA®; it is a technology that enables users to solve many complex problems on their GPU cards. It is a parallel computing platform and application programming interface (API). CUDA® consists of a parallel computing architecture and developer tools, libraries, and programming directives for GPU computing. The CUDA® platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements for the execution of compute kernels [17]. The Kernel is a code written for execution on the GPU. Kernels are functions that can run on a large number of threads. The parallelism arises from each thread independently running the same program on different data [17].

III. Admittance Matrix Formation

Building the network matrices is a burdensome computational procedure as obtaining impedance or admittance matrices involves inversions and multiplications. The admittance matrix is formed from the electrical parameters of the power system components (e.g. transmission lines, transformers, etc.). Several approaches are available for the formation of the admittance matrix; it includes using the element-by-element approach (as described in many power systems analysis textbooks [18], [19], etc.), more computer-oriented approaches take advantages of using the incidence [20] and connectivity matrix [21], [22]. In this section, a brief introduction to the admittance matrix formation is presented. The element-by-element method is initially presented, and then a full matrix method using the connection matrix is presented.

A. Nodal Admittance Matrix ($\mathbf{Y_{bus}}$)

The nodal admittance matrix termed the admittance matrix, $\mathbf{Y_{bus}}$, is a square $N_{bus}\times N_{bus}$ matrix describing the nodal admittances of a N_{bus} buses power system [9]. The $\mathbf{Y_{bus}}$ is a heavily weighted matrix in the main diagonal and is highly sparse; this is caused by the topological properties of the power system, as it is not fully meshed (complete graph) [13]. Also, the $\mathbf{Y_{bus}}$ is symmetric in terms of the principal diagonal element is negated admittance, unless mutual coupling elements are included. Considering a symmetrical and balanced power system with N_{bus} buses, the positive sequence network model, including all constant impedance elements can be represented by the complex admittance matrix ($\mathbf{Y_{bus}}$). The $N_{bus}\times N_{bus}$ admittance matrix ($\mathbf{Y_{bus}}$) relates the complex current injections ($\mathbf{I_{bus}}$) and the complex voltages ($\mathbf{V_{bus}}$) at the buses :

$$\mathbf{I_{bus}} = \mathbf{Y_{bus}}\mathbf{V_{bus}} \tag{1}$$

B. Formation of $\mathbf{Y_{bus}}$ Element by Element

Assuming the admittance of each branch element of the power system is known in the form: $y_{ik} = g_{ik} + jb_{ik}$. where y_{ik} represents the actual admittance of the power device connecting the busbar 'i' and 'k', g_{ik} and b_{ik} are the real (conductance) and imaginary (susceptance) components of y_{ik} respectively. The individual elements of the matrix ($\mathbf{Y_{bus}}$) are formulated using straightforward rules. The elements of the main diagonal (Y_{ii}), also known as self-admittances, are calculated by the sum of the admittance physically connected to the bus [18], [20]:

$$Y_{ii} = \sum_{i=1}^{N} \hat{Y}_{ii} \tag{2}$$

The off-diagonal elements (Y_{ij}), also known as mutual-admittances, are calculated by the negative branch admittances connected directly between the corresponding busses.

$$Y_{ij} = -\hat{Y}_{ij} \tag{3}$$

Note that the $\mathbf{Y_{bus}}$ matrix is symmetric unless there are branches whose admittance is direction-dependent.

C. Formation of $\mathbf{Y_{bus}}$ Matrix, Matrix approach

Considering a symmetrical and balanced power system with N_{bus} buses, the network model, including all constant impedance elements, can be represented by the complex admittance matrix ($\mathbf{Y_{bus}}$). The $N_{bus}\times N_{bus}$ admittance matrix related the complex current injections at the buses ($\mathbf{I_{bus}}$) and the complex voltages ($\mathbf{V_{bus}}$):

$$\mathbf{I}_{\mathbf{bus}} = \mathbf{Y}_{\mathbf{bus}}\mathbf{V}_{\mathbf{bus}} \tag{4}$$

the admittance matrix is expressed in terms of the connection matrixes:

$$\mathbf{Y}_{\mathbf{bus}} = \left[\mathbf{C}_{\mathbf{from}}\right]^{\mathbf{T}}\mathbf{Y}_{\mathbf{from}} + \left[\mathbf{C}_{\mathbf{to}}\right]^{\mathbf{T}}\mathbf{Y}_{\mathbf{to}} + \mathbf{Y}_{\mathbf{shunt}} \tag{5}$$

where:

$$\begin{cases} \mathbf{Y}_{\mathbf{from}} = \mathbf{Y}_{\mathbf{from,from}}\mathbf{C}_{\mathbf{from}} + \mathbf{Y}_{\mathbf{from,to}}\mathbf{C}_{\mathbf{to}} \\ \mathbf{Y}_{\mathbf{to}} = \mathbf{Y}_{\mathbf{to,from}}\mathbf{C}_{\mathbf{from}} + \mathbf{Y}_{\mathbf{to,to}}\mathbf{C}_{\mathbf{to}} \end{cases} \tag{6}$$

$$\begin{bmatrix} \mathbf{Y}_{\mathbf{from}} \\ \mathbf{Y}_{\mathbf{to}} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{\mathbf{from,from}} & \mathbf{Y}_{\mathbf{from,to}} \\ \mathbf{Y}_{\mathbf{to,from}} & \mathbf{Y}_{\mathbf{to,to}} \end{bmatrix} \begin{bmatrix} \mathbf{C}_{\mathbf{from}} \\ \mathbf{C}_{\mathbf{to}} \end{bmatrix}$$

where $\mathbf{C}_{\mathbf{from}}$ and $\mathbf{C}_{\mathbf{to}}$ are the connection matrices providing the connectivity of the network elements. The (i,j)th element of of $\mathbf{C}_{\mathbf{from}}$ and the (i,k) element of $\mathbf{C}_{\mathbf{to}}$ are equal to 1 for each branch i, where the branch i connects from bus j to bus k, any other element in $\mathbf{C}_{\mathbf{from}}$ and $\mathbf{C}_{\mathbf{to}}$ are zero. The vector $\mathbf{Y}_{\mathbf{shunt}}$ represents the shunt admittance of all buses $\mathbf{Y}_{\mathbf{shunt}} = \mathbf{G}_{\mathbf{shunt}} + j\mathbf{B}_{\mathbf{shunt}}$.

D. Sparsity in Electrical Power Systems

Electrical power systems are particular in terms of topology; the transmission system is often highly interconnected, creating a mesh system, but the distribution system is mainly a radial connection. The lack of massive connectivity between nodes makes the admittance matrix populated with many zeros; in other words, it is a sparse matrix which contains very few non-zero elements compared to the zero elements. Therefore, the use of dense matrix representation (2-dimensional array) in power system calculation when the admittance matrix is involved becomes inefficient, as memory is used to store many zeros. An efficient alternative to treat the admittance matrix of real power systems (especially huge ones) is the use of sparse matrix representation. The sparse matrix representation allows saving memory used by storing only the non-zero entries. There are several mechanisms to represents sparse matrixes, and they depend on the number and distribution of the non-zero entries. Depending on the sparse matrix representation (data structures), it can yield considerable savings in memory when compared to the dense matrix representation (2-dimensional array). For instance, a 10,000×10,000 identity matrix might require 800 MB memory if the matrix is stored considering the full array; but considering sparse representation in the memory storage, only approximately 0.25 MB of memory is needed. The advantages of memory management are apparent. The data structure used in the sparse matrix can be divided into two groups depending on the purpose: (i) Support efficient modification: DOK (Dictionary of keys), LIL (List of lists), or COO (Coordinate list). (ii) Support efficient access and matrix operations, such as CSR (Compressed Sparse Row) or CSC (Compressed Sparse Column) [23].

IV. Experimental Results

This paper focuses on a multi-core CPU implementation of two methods of forming the admittance matrix presented in Sections IIIb and IIIc (i.e. multi-core GPU is not considered). The algorithms are implemented by the authors using MATLAB™ R2020a and taking advantage of Parallel Computing Toolbox. The performance of implementations in each multi-core platform is based on the total calculation time. In this paper, we are not interested in measuring the first-time cost; as a consequence, the time of running the code a second time is taken into consideration. The authors are aware of the influence on the computational time of other processes in the computation in and operating system like Microsoft Windows, as a consequence, the authors have minimised (in a practical fashion) the number of processes running during the simulations (e.g. reducing the number of start-up processes, background running processes, third-party software service) at the time of ensuring the stability of the system. Also, the authors are running a set of experiments to present a probabilistic analysis of multiple runs, but such results will be included in a future publication.

A. Multi-Core Plattforms

Four target platforms consisting of a multi-core CPU are used for testing purposes in this paper; details are presented in Table I. The selected platforms include CPU with 6 and 12 independent central processing units, including a wide range of fast memory located in the processor, CPU cache. The processors considered in this assessment are equipped with 12 and 24 threads.

B. Test Systems

In this paper, the main interest is to evaluate the performance of several software/hardware approaches to solve the problem of creating the admittance matrix, considering massive power systems. As a result, the research team decided to use very well-recognised and used synthetic electric grid models. The research team at the Smart Grid Centre within the Texas A&M Engineering Experiment Station (TEES) has created an excellent repository of electric grid test cases [24], [25]; those test cases are synthetically created with a fictitious representation of the power system, but they were designed and created to statistically and functionally be similar to actual electric power systems and employed confidential critical energy infrastructure information (CEII).

TABLE I. Summary of the Main Characteristics of the Hardware used for The Experiments

Platform	*1*	*2*	*3*	*4*
Code name	*Black*	*Strix*	*Super*	*Mega*
Processor	Inter® Core® i7 -8850H	Intel® Core® i7-10750H	AMD Ryzen™ 9 3900X	Intel® Xeon® W-3235
Processor Base Frequency	2.60 GHz	2.60 GHz	3.80 GHz	3.30 GHz
CPU Cache	9 MB	12 MB	6 MB	19.25 MB
#Cores	6	6	12	12
# of Threads	12	12	24	24

TABLE II. Summary of the Main Characteristics of the Test Systems used for The Experiments

Test System Name	N_{bus}	*Representative*
ACTIVSg200	200	Central Illinois
ACTIVSg500	500	South Carolina
ACTIVSg2000	2000	Texas
ACTIVSg10k	10,000	Western USA
ACTIVSg25k	25,000	North-eastern USA
ACTIVSg70k	70,000	Eastern USA

The sparsity is an intrinsic feature of the admittance matrix ($\mathbf{Y}_{\mathbf{bus}}$). The sparsity is present in the admittance matrix because of the absence of certain interconnections; the normal power system is not a fully connected graph. Mathematically speaking, the sparsity of the admittance matrix is defined as:

$$Sparcity = \frac{N_z}{\left[N_{bus}\right]^2} \times 100\% \quad (7)$$

where N_z represents the total number of zero elements in the admittance $N_{bus} \times N_{bus}$ matrix. Typical power systems have a highly sparse admittance matrix with sparsity as high as 97%. For instance, the test system ACTIVSg70k, N_{bus} = 70,000 buses, a total of 4.9×10^9 elements, N_z = 236,636 are not zero, as a consequence, the sparsity of the admittance matrix results: 99.99517%.

C. Experiments and results

The multi-core cross-platform assessment includes two main experiments: (i) $\mathbf{Y_{bus}}$ formation element by element (section IIIb), and (ii) Matrix formation (section IIIc) called from here onward 'Optimised Code'. The input data is the same in both cases; it uses a data structure having the connectivity and individual series impedances, as a consequence, the assessment of simulation time is based on the same input data and the total time of simulation measures the time used by the admittance matrix algorithm. Space and dense matrix representation are considered for each experiment, and parallel/non-parallel computing are considered.

TABLE III. ADMITTANCE MATRIX (Y_{BUS}) ELEMENT BY ELEMENT COMPUTATIONAL TIME RESULTS ON PLATFORM 1- CODENAME "BLACK"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.03401	0.02456	0.01505	0.01388
500	0.14337	0.14180	0.09782	0.08635
2k	2.84872	3.14546	1.83264	1.78103
10k	54.11942	57.37445	35.82635	35.37565
25k	341.13466	383.26233	240.27718	NO
70k	2649.34057	2821.18607	NO	NO

TABLE IV. ADMITTANCE MATRIX (Y_{BUS}) ELEMENT BY ELEMENT COMPUTATIONAL TIME RESULTS ON PLATFORM 2- CODENAME "STRIX"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.0318	0.02131	0.01161	0.01193
500	0.1257	0.12382	0.07042	0.07002
2k	2.5741	2.50483	1.44564	1.44805
10k	50.8096	49.82783	31.96384	31.81366
25k	323.1769	319.76153	213.93833	206.58091
70k	2443.8004154	2760.64439	NO	NO

A total of 384 simulations were performed in this paper (two simulations per case, the second one was recorded to avoid tendencies caused by the first-time cost). Numerical results of all experiments are shown in Tables III to X. Overall results show the $\mathbf{Y_{bus}}$ formation element by element results in the worst algorithm when compared with the matrix approach; it is right for all multi-core platforms and implementations: parallel or not and sparse/dense. Dense matrix approach shows better performance for a smaller system; however, the memory requirements of the ACTIVSg70k makes it impossible to run dense matrix calculation across all platforms. Finally, parallel computing makes the calculations faster for all the platforms. Codename 'Super' and 'Mega' were the fastest multi-core platforms, the higher CPU clock and number of threads makes this configuration to run fast calculation considering the element by element approach.

TABLE V. ADMITTANCE MATRIX (Y_{BUS}) ELEMENT BY ELEMENT COMPUTATIONAL TIME RESULTS ON PLATFORM 4- CODENAME "SUPER"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.02653	0.02264	0.01053	0.01073
500	0.13090	0.13032	0.06877	0.06803
2k	2.81127	2.77271	1.48254	1.49980
10k	53.39978	51.16918	31.66811	31.41345
25k	329.74340	336.42700	203.42113	211.42329
70k	2566.97005	2548.96518	NO	NO

TABLE VI. ADMITTANCE MATRIX (Y_{BUS}) ELEMENT BY ELEMENT COMPUTATIONAL TIME RESULTS ON PLATFORM 4- CODENAME "MEGA"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU No Dense*
200	0.02214	0.02218	0.01248	0.01180
500	0.12492	0.12393	0.07336	0.07313
2k	2.5723	2.59147	1.54307	1.56215
10k	50.43169	51.06726	33.31061	33.87128
25k	319.33035	326.55104	214.01856	217.42571
70k	2456.43942	2486.31923	NO	NO

TABLE VII. ADMITTANCE MATRIX (Y_{BUS}) OPTIMIZED CODE COMPUTATIONAL TIME RESULTS ON PLATFORM 1- CODENAME "BLACK"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.00022	0.00024	0.00377	0.00440
500	0.00049	0.00055	0.05489	0.05771
2k	0.00158	0.00147	2.93238	3.08360
10k	0.00499	0.00556	300.08864	287.90158
25k	0.01493	0.01518	NO	NO
70k	0.04864	0.05230	NO	NO

TABLE VIII. ADMITTANCE MATRIX (Y_{BUS}) OPTIMIZED CODE COMPUTATIONAL TIME RESULTS ON PLATFORM 2- CODENAME "STRIX"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.00029	0.00050	0.00376	0.00378
500	0.00050	0.00041	0.03890	0.04507
2k	0.00168	0.00122	2.84101	2.78176
10k	0.00570	0.00441	254.28606	251.90239
25k	0.01507	0.01158	NO	NO
70k	0.04987	0.03769	NO	NO

TABLE IX. ADMITTANCE MATRIX (Y_{BUS}) OPTIMIZED CODE COMPUTATIONAL TIME RESULTS ON PLATFORM 3- CODENAME "SUPER"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.00022	0.00021	0.00020	0.0021
500	0.00042	0.00044	0.0038	0.0035
2k	0.00145	0.00141	0.0143	0.0163
10k	0.00453	0.00447	0.00495	0.00489
25k	0.01162	0.01144	0.01290	0.01270
70k	0.03936	0.03925	0.04301	0.04330

TABLE X. ADMITTANCE MATRIX (Y_{BUS}) OPTIMIZED CODE COMPUTATIONAL TIME RESULTS ON PLATFORM 3- CODENAME "MEGA"

Test System	*No parallel CPU only Sparse*	*Parallel CPU only Sparse*	*No parallel CPU only Dense*	*Parallel CPU Dense*
200	0.00026	0.00026	0.00025	0.00026
500	0.00078	0.00074	0.00046	0.00046
2k	0.00147	0.00167	0.00142	0.00161
10k	0.00477	0.00490	0.00488	0.00478
25k	0.01247	0.01869	0.01279	0.01251
70k	0.04419	0.04316	0.04291	0.04317

The numerical results show the use of parallel computing is marginally better than using the non-parallel solution. However, the marginal difference is present in almost all the assessed cases as a consequence, the parallel computing looks like a better option (if possible). The matrix formation approach to building the $\mathbf{Y_{bus}}$ makes the calculation less demanding when comparing computing times. The use of a

dense matrix representation was not possible in codenames "Black" and "Strix" as these approaches put the burden on the RAM requirements. The use of sparse matrix (parallel and non-parallel calculations) resulted in the fastest method, and it is basically attributable to two elements: (1) MATLAB is a numerical computing environment which uses a proprietary programming language with particular emphasis on mathematical calculations, and matrix approach, (2) the matrix approach is already a more efficient way of using and accessing memory that speeds up calculations with using sparse representation.

V. Conclusions

This paper presents the fundamentals of two approaches to calculate the admittance matrix of electrical power systems. The computational time is used to compare the performance of multi-core platforms, and four different computing systems are used to calculate the admittance matrix of six tests system, algorithm implementation has included sparse/dense matrix representation of $\mathbf{Y}_{\mathbf{bus}}$ and parallel/non-parallel computing. The input data for both algorithms was the same; it is based on the standard connectivity and series admittance description used in many power system analysis software and the data formation time is not part of the computational time. The element by element exhibit the worst computational time performance, the nature of the approach requires searching and element inside the power system structure to be added into the admittance matrix. Finding an individual admittance element inside a data structure of the power system requires is a computational consuming process, the admittance matrix program instruct the CPU a request for a specific element inside the memory, but searching the memory location of that piece of information requires more time than the time required to move a whole allocated data in the memory, the lasted approach is used by the matrix approach and makes it more efficient in computational time. Overall results show that the $\mathbf{Y}_{\mathbf{bus}}$ formation with the element-by-element approach results in the worst algorithm when compared with the matrix approach. Parallel computing implementation tends to produce slightly faster results when compared within selected multi-core platforms.

References

[1] A. Perilla, J. L. Rueda Torres, S. Papadakis, E. Rakhshani, M. van der Meijden, and F. Gonzalez-Longatt, "Power-Angle Modulation Controller to Support Transient Stability of Power Systems Dominated by Power Electronic Interfaced Wind Generation," *Energies*, vol. 13, no. 12, p. 3178, Jun. 2020, doi: 10.3390/en13123178.

[2] F. S. Gorostiza and F. Gonzalez-Longatt, "Deep Reinforcement Learning-Based Controller for SOC Management of Multi-Electrical Energy Storage System," *IEEE Trans. Smart Grid*, pp. 1–1, 2020, doi: 10.1109/TSG.2020.2996274.

[3] C. Adiyabazar, M. N. Acosta, F. Gonzalez-Longatt, J. L. Rueda, and P. Palensky, "Assessment of Under-Frequency Load Shedding in Mongolia Considering Inertia Scenarios," in *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*, Jun. 2020, pp. 1256–1261, doi: 10.1109/ISIE45063.2020.9152584.

[4] M. N. Acosta, D. Pettersen, F. Gonzalez-Longatt, J. Peredo Argos, and M. A. Andrade, "Optimal Frequency Support of Variable-Speed Hydropower Plants at Telemark and Vestfold, Norway: Future Scenarios of Nordic Power System," *Energies*, vol. 13, no. 13, p. 3377, Jul. 2020, doi: 10.3390/en13133377.

[5] IEEE, *IEEE Std 399. Recommended Practice for Industrial and Commercial Power Systems Analysis (Brown Book)*, vol. 1995, no. Lcc. 1998.

[6] A. Peña Asensio, F. Gonzalez-Longatt, S. Arnaltes, and J. L. Rodríguez-Amenedo, "Analysis of the Converter Synchronizing Method for the Contribution of Battery Energy Storage Systems to Inertia Emulation," *Energies*, vol. 13, no. 6, p. 1478, Mar. 2020, doi: 10.3390/en13061478.

[7] A. Perilla, S. Papadakis, J. L. Rueda Torres, M. van der Meijden, P. Palensky, and F. Gonzalez-Longatt, "Transient Stability Performance of Power Systems with High Share of Wind Generators Equipped with Power-Angle Modulation Controllers or Fast Local Voltage Controllers," *Energies*, vol. 13, no. 16, p. 4205, Aug. 2020, doi: 10.3390/en13164205.

[8] H. Chamorro, F. Gonzalez, K. Rouzbehi, R. Sevilla, H. Chavez, and V. Sood, "Innovative Primary Frequency Control in Low-Inertia Power Systems Based on Wide-Area RoCoF Sharing," *IET Energy Syst. Integr.*, Feb. 2020, doi: 10.1049/iet-esi.2020.0001.

[9] G. W. Stagg and A. H. El-Abiad, *Computer methods in power system analysis*. New York: McGraw-Hill, 1968.

[10] T. Krechel, F. Sanchez, F. Gonzalez-Longatt, H. Chamorro, and J. L. Rueda, "A Transmission System Friendly Micro-grid: Optimising Active Power Losses," 2019.

[11] M. N. Acosta, F. Gonzalez-Longatt, S.Denysiuk, and H. Strelkova, "Optimal settings of Fast Active Power Controllers: Nordic Case," 2020.

[12] E. Rakhshani, A. Perilla, J. R. Torres, F. G. Longatt, T. B. Soeiro, and M. Van Der Meijden, "FAPI Controller for Frequency Support in Low-Inertia Power Systems," *IEEE Open Access J. Power Energy*, pp. 1–1, 2020, doi: 10.1109/OAJPE.2020.3010224.

[13] W. F. Tinney and W. S. Meyer, "Solution of Large Sparse Systems by Ordered Triangular Factorisation," *IEEE Trans. Automat. Contr.*, vol. 18, no. 4, pp. 333–346, 1973, doi: 10.1109/TAC.1973.1100352.

[14] F. Gonzalez-Longatt, J. L. Rueda, and D. Bogdanov, "Assessment of the Critical Clearing Time in Low Rotational Inertia Power Systems," in *2018 20th International Symposium on Electrical Apparatus and Technologies (SIELA)*, Jun. 2018, pp. 1–4, doi: 10.1109/SIELA.2018.8447128.

[15] R. Idema, D. J. P. Lahaye, C. Vuik, and L. Van Der Sluis, "Scalable Newton-Krylov solver for very large power flow problems," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 390–396, Feb. 2012, doi: 10.1109/TPWRS.2011.2165860.

[16] Z. Li, V. D. Donde, J. C. Tournier, and F. Yang, "On limitations of traditional multi-core and potential of many-core processing architectures for sparse linear solvers used in large-scale power system applications," 2011, doi: 10.1109/PES.2011.6039675.

[17] "What is GPGPU? Definition and FAQs | OmniSci." https://www.omnisci.com/technical-glossary/gpgpu (accessed Aug. 30, 2020).

[18] J. J. Grainger and W. D. Stevenson, *Power system analysis*. New York ; London: McGraw-Hill, 1994.

[19] H. Saadat, *Power system analysis*, 3rd ed. [United States]: PSA Pub., 2010.

[20] H. E. Brown, *Solution of large networks by matrix methods*, 2nd ed. New York ; Chichester: Wiley, 1985.

[21] J. Arrillaga and C. P. Arnold, *Computer analysis of power systems*. Chichester, England ; New York: Wiley, 1990.

[22] F. Milano, *Power system modelling and scripting*, 1st ed. New York: Springer, 2010.

[23] "Sparse matrices (scipy.sparse) — SciPy v1.5.2 Reference Guide." https://docs.scipy.org/doc/scipy/reference/sparse.html (accessed Aug. 30, 2020).

[24] "Electric Grid Test Cases." https://electricgrids.engr.tamu.edu/electric-grid-test-cases/ (accessed Jul. 23, 2020).

[25] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Grid Structural Characteristics as Validation Criteria for Synthetic Networks," *IEEE Trans. Power Syst.*, vol. 32, no. 4, pp. 3258–3265, Jul. 2017, doi: 10.1109/TPWRS.2016.2616385.