IET Generation, Transmission & Distribution

Research Article

Reduce–factor–solve for fast Thevenin impedance computation and network reduction

ISSN 1751-8687
Received on 20th April 2018
Revised 5th October 2018
Accepted on 30th October 2018
E-First on 18th December 2018
doi: 10.1049/iet-gtd.2018.5330
www.ietdl.org

Stefan Sommer[1] ✉, *Andreas Aabrandt*[1], *Hjörtur Jóhannsson*[2]
[1]*Department of Computer Science, University of Copenhagen, Denmark*
[2]*Department of Electrical Engineering, Technical University of Denmark, Denmark*
✉ *E-mail: sommer@di.ku.dk*

Abstract: The complexity and volatility of power system operation increase when larger parts of the power production is based on distributed and non-controllable renewable energy sources. Ensuring stable and secure operation becomes more difficult in these modern power systems. For security assessment, the results of traditional offline simulations may become obsolete prior to the completion of the assessment. In contrast, real-time stability and security assessment aims at *online* computation, and it is therefore dependent on very fast computation of properties of the grid operating state. The study develops the *reduce–factor–solve* approach to real-time computation of two key components in real-time assessment methods, network reduction, and calculation of Thevenin impedances. The aim is to allow online stability assessment for very complex networks. The theoretical foundation behind the reduce–factor–solve approach is described together with the ability to handle both algorithms in a common framework. By exploiting parallelisation of the reduce and solve steps in combination with fast matrix factorisation, Thevenin impedances and reduced networks are computed much faster than previous approaches. The reduce–factor–solve algorithm is evaluated on power grids of varying complexity to show that Thevenin impedance computation and network reduction for complex power systems can be performed on a milliseconds time scale.

1 Introduction

Although traditional power systems are characterised by centralised controllable energy production, efforts on decarbonising the power system often lead to the distributed power production based on non-controllable renewable energy sources. This general shift from low numbers of high-power units to high numbers of low-power units with fluctuating production creates new challenges for stable and secure operation.

In traditional power systems, stability and security could be assessed offline and sensitivities to various contingencies established by running time-consuming simulations. Multiple factors of the future power system can challenge this approach: the complexity of a power grid is greater when the level of decentralisation is high. This results in higher computational requirements and thus longer runtime for simulations. The power system should still be able to operate under rapidly changing conditions, e.g. when including weather dependent energy sources. Large fluctuations of the system operating point can be common, and in combination these factors may make the results of conventional offline stability assessment obsolete prior to the completion of time domain simulation. From a stability assessment point of view, the answer to these complications lies in faster assessment methods [1]. Speed-up of assessment algorithms can be obtained by developing new assessment methods or by reducing the execution time of critical elements of already existing methods. However, no matter which strategy is chosen, fast algorithms for extracting properties of the grid operating state are needed.

This paper presents a general approach to fast computation of two critical parts of important assessment methods: calculating the system Thevenin impedances and reducing the network to the generator or voltage controlled nodes (*vcs*) only. The *reduce–factor–solve* approach allows online computation of both operations in a common framework. As several approaches to real-time stability assessment require knowledge of Thevenin impedances and properties of the reduced network, the developed algorithms make the use of these approaches feasible for complex power systems at smaller time resolutions. As we validate experimentally, the presented approach allows fast Thevenin impedance computation on networks with close to 8000 buses on a millisecond scale. The paper shows how the Thevenin impedance computation constitute a particular case of network reduction, and this property is used to cover both operations with the reduce–factor–solve approach. The approach as illustrated in Fig. 1 is identified as a mix of left-looking and frontal matrix factorisation algorithms, and the division between serial and highly parallel parts is used to reduce the runtime of both Thevenin impedance computations and network reduction.

The work extends the conference paper [2] by including full network reduction and node-elimination as a core part of the algorithm resulting in the *reduce–factor–solve* framework. While node elimination is generally non-parallelizable, we describe here how fine-grained parallelisation in the elimination procedure can be exploited, and how graphics processing units (GPUs) can be employed to speed-up part of the network reduction task.

The paper starts with background information on real-time stability assessment and computational issues before progressing to discussing Thevenin impedances and previous algorithms. It is shown how the voltage controlled part of the network gives rise to a dense submatrix, and this is used to develop the reduce–factor–solve approach. The role of node elimination is discussed before covering full network reduction. In Section 8, parallelisation and implementation details are discussed, and the performance and validity of the reduce–factor–solve approach is evaluated in Section 9. The paper ends with description of future work and concluding remarks.

2 Background

Thevenin impedance calculations constitute a major component of important stability assessment methods. For example, in [3], a method is proposed for real-time assessment of rotor angle stability which exploits analytically derived expressions for critical stability boundaries [4]. The Thevenin impedance seen from each node of constant voltage is used to determine the distance to a stability boundary of each synchronous machine, and the voltage stability monitoring is based on local measurements and Thevenin impedances in [5–8]. Thevenin impedance based assessment were

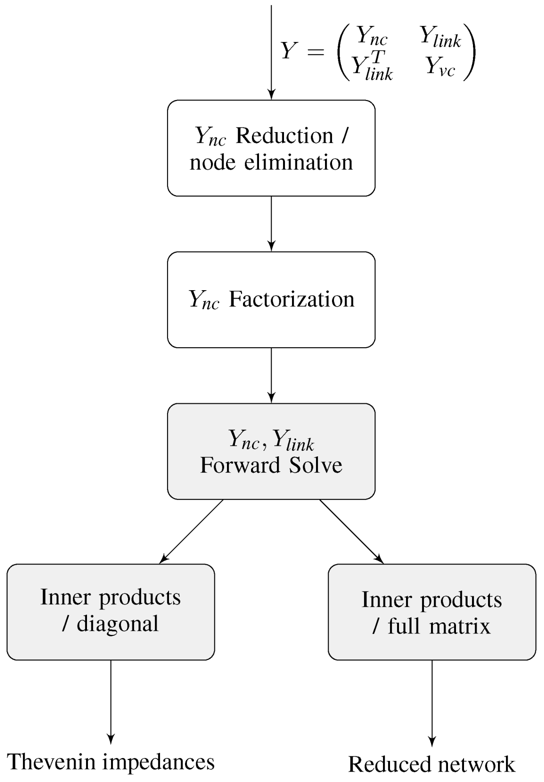
$Y = \begin{pmatrix} Y_{nc} & Y_{link} \\ Y_{link}^T & Y_{vc} \end{pmatrix}$
Y_{nc} Reduction / node elimination
Y_{nc} Factorization
Y_{nc}, Y_{link} Forward Solve
Inner products / diagonal
Inner products / full matrix
Thevenin impedances
Reduced network

Fig. 1 *Reduce–factor–solve approach, see also Algorithm 1. The full system admittance matrix* $\boldsymbol{Y}$ *is partitioned into the voltage controlled* $\boldsymbol{Y}_{\rm vc}$*, non-voltage controlled* $\boldsymbol{Y}_{\rm nc}$*, and link* $\boldsymbol{Y}_{\rm link}$ *parts. The algorithm performs a fill-in limiting reduction/node elimination followed by a factorisation of* $\boldsymbol{Y}_{\rm nc}$*. Computation of Thevenin impedances (left) or fully reduced network (right) follows after a forward solve and inner product computations. The grayed boxes represent fully parallelizable parts of the algorithm, while the white boxes are primarily serial allowing only fine-grained parallelisation*

proposed in [9–11], and approaches for real-time static security assessment exploiting coupled Thevenin equivalents are reported in [12–14]. Online stability assessment with the above methods is dependent on fast Thevenin impedance computation, and the algorithm presented here therefore addresses a real issue: without fast algorithms, online stability assessment will be limited in the time resolution and in the size of the network concerned. To the best of our knowledge, no method that focuses on computational aspects of Thevenin impedance computation is described in the literature besides [2, 3].

Direct methods for transient stability using Lyaponov functions [15] compare the energy of the grid operating point against the value of the system in critical states. Each generator contributes to the energy and the evaluation of the energy function is therefore dependent on the transfer conductances between all pairs of generators [16]. The transfer conductances constitute the elements of the admittance matrix of the network reduced to the generator nodes only. The fast *network reduction* method developed here will therefore allow fast evaluation of energy functions with the Lyaponov method. Similarly, assessment methods based on the extended equal area criterion [17] rely on the admittance matrix of the reduced network for calculation of the acceleration criterion and identification of critical machines. See [18] for mathematical aspects of network reduction and [19] for step-by-step matrix reduction closely related to node elimination as discussed here.

To minimize the execution time of computations involving power grids, sparsity of the network matrices is often exploited. Sparse algorithms are, however, often hard to parallelize, and a trade-off between exploiting sparsity and parallelisation is often necessary. Here, the high level of parallelism of the *solve*-step in the developed network reduction algorithm allows for fast evaluation of the energy in Lyaponovs direct method. In contrast, when solutions to linear systems are needed, e.g. for implicit time simulation with the Lyaponov direct method, the larger but more sparse non-reduced system should be used [20]. Additional typical network computations involve sparse matrix factorisation [21] and relaxation methods [22]. See also [23] for comparison of factorisation and relaxation methods.

3 Power system and representation

An arbitrary power grid consisting of N nodes (buses) is considered throughout the paper. In the grid, the steady state voltage magnitude at $M \le N$ nodes is kept constant by means of voltage control. These M nodes include the nodes representing synchronous generators' internal voltage (behind $X_{\rm d}$) if they are manually excited. Let $\boldsymbol{Y}$ denotes the system admittance matrix, the system node voltage equation is $I = \boldsymbol{Y}V$. The M *vc*s and the $N - M$ nodes of non-controlled voltage (*nc*s) can be ordered so that the *nc*s and *vc*s are numbered by indices $1, \ldots, N - M$ and $N - M + 1, \ldots, N$, respectively. The system admittance matrix then takes the form

$$\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{Y}_{\rm nc} & \boldsymbol{Y}_{\rm link} \\ \boldsymbol{Y}_{\rm link}^T & \boldsymbol{Y}_{\rm vc} \end{pmatrix} \tag{1}$$

where $\boldsymbol{Y}_{\rm nc}$ denotes the admittance matrix of only the *nc* part of the system, $\boldsymbol{Y}_{\rm vc}$ denotes the admittance matrix of the *vc* part, and $\boldsymbol{Y}_{\rm link}$ encodes the links between the *nc* and *vc* parts of the network.

4 Transfer admittances and Thevenin impedances

This section describes two mathematical formulations of the system Thevenin impedances as seen from the voltage controlled *vc* nodes. For each *vc* node k, the aim is to compute the *Thevenin impedance* for the node, i.e. the impedance seen from node k when all *vc* nodes besides node k are shorted. This situation can be modeled by removing all *vc* nodes besides k from the system, and the Thevenin impedance $Z_{{\rm th},k}$ can then be obtained as the last diagonal element of inverse of the resulting admittance matrix.

4.1 Thevenin impedances from LU-factorisation

It is shown in [24] that the Thevenin impedances $Z_{{\rm th},k}$ can be obtained from an $\boldsymbol{LU}$-factorisation of the system admittance matrix. The $\boldsymbol{LU}$-factorisation [25] splits a matrix into a product of a lower diagonal and an upper diagonal matrix, e.g. the admittance matrix $\boldsymbol{Y}$ is factorised into the product $\boldsymbol{Y} = \boldsymbol{LU}$. Using the factorisation, the impedance $Z_{{\rm th},k}$ can be found by the formula

$$Z_{{\rm th},k}^{-1} = \boldsymbol{Y}_{(k,k)} - \hat{\boldsymbol{L}}_{k\cdot}\hat{\boldsymbol{U}}_{\cdot k} \tag{2}$$

with the last term being the inner product between the entries $1, \ldots, N - M$ of the kth row of the matrix $\boldsymbol{L}$ and of the kth column of the matrix $\boldsymbol{U}$.

4.2 Schur complement formulation

The Thevenin impedances can equivalently be expressed in terms of the Schur complement of the voltage controlled part of the admittance matrix. With system loads represented by their admittance values, no current enters the *nc*s, and the network equation can be stated as

$$\begin{pmatrix} 0 \\ I_{\rm vc} \end{pmatrix} = \begin{pmatrix} \boldsymbol{Y}_{\rm nc} & \boldsymbol{Y}_{\rm link} \\ \boldsymbol{Y}_{\rm link}^T & \boldsymbol{Y}_{\rm vc} \end{pmatrix} \begin{pmatrix} V_{\rm nc} \\ V_{\rm vc} \end{pmatrix}. \tag{3}$$

Using the *Schur complement* [18, 26]

$$S = \boldsymbol{Y}_{\rm vc} - \boldsymbol{Y}_{\rm link}^T \boldsymbol{Y}_{\rm nc}^{-1} \boldsymbol{Y}_{\rm link} \tag{4}$$

of $\boldsymbol{Y}_{\rm vc}$, the *vc*-part of the solution to (3) can be obtained from the reduced system $I_{\rm vc} = SV_{\rm vc}$. Looking only at the diagonal elements

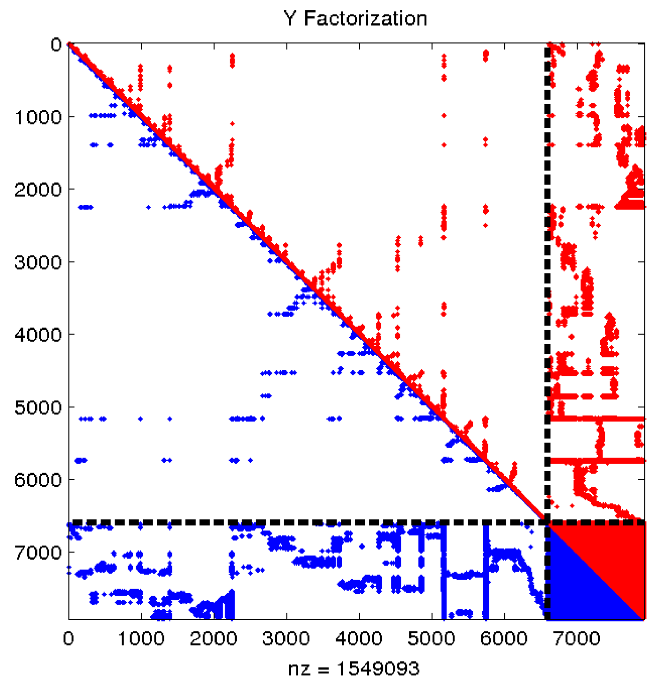

Fig. 2 *Sparsity patterns of the **LU**-factorisation of the full admittance matrix **Y**. The excessive fill-in visible in the lower right vc-part limits the performance of the algorithm. In contrast, factorisation of the nc-submatrix only with the reduce–factor–solve approach can be done with very limited fill-in and consequently very fast factorisation. The reduce–factor–solve algorithm reduces both the dimension of the matrix to be factored (in this case from 7917 × 7917 to 960 ×960) and the number of non-zeros in the factors (from 1,549,093 to 10,174)*

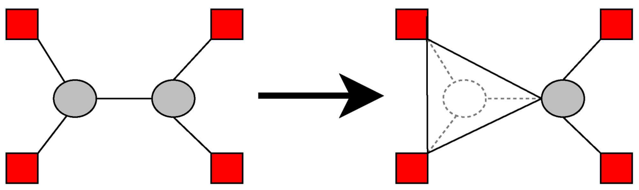

Fig. 3 *Elimination of one of two interior nodes in a six node network. The node to be eliminated has degree three and with three new branches added to the reduced network, the total number of branches is kept constant. This preserves the sparsity. Elimination of nodes with higher degree will result in an increased number of branches reducing sparsity*

$S_{(d,d)}$, $d = 1, \dots, M$, the right-hand side of (2) equals $S_{(d,d)}$ with $k = d + N - M$. The Thevenin impedances are therefore the element-wise inverses of the diagonal of the Schur complement S.

5 Approaches to computing Thevenin impedances

In [24], $\boldsymbol{LU}$-factorisation of the full system admittance matrix $\boldsymbol{Y}$ and (2) is used for computing Thevenin impedances. The benefit of using this approach is that only one factorisation of the full admittance matrix is needed in order to compute $Z_{\text{th},k}$ for *all vc* nodes. This method is analysed below in order to show how a faster approach can be developed, and the method is used as basis for the comparisons in Section 9.

5.1 Complexity and fill-in

Due to the very high sparsity of network matrices, $\boldsymbol{LU}$-factorisation is in general a very efficient procedure. Though the worst case performance scales cubicly in the number of nodes, i.e. performance is $O(N^3)$ in worst case, the complexity will be linear in most practical cases, see [23]. A key factor in achieving this complexity is minimising the number of *fill-ins*, non-zero elements of the factors $\boldsymbol{L}$ and $\boldsymbol{U}$ that are not present in $\boldsymbol{Y}$. The number of fill-ins is highly dependent on the ordering of the matrix $\boldsymbol{Y}$. For network matrices, ordering algorithms like approximated minimum degree (AMD, [27]) and variants ensure a very low degree of fill-in. The number of both non-zeros in $\boldsymbol{Y}$ and additional fill-ins are in practice close to linearly correlated with N, implying that the factorisation will be close to linear in complexity.

In (1), an ordering with the *nc*s occurring with lower indices than the *vc*s is used. This ordering is required for (2) that allows us to extract the Thevenin impedances. To adhere to this indexing convention, [24] applies AMD ordering to the submatrices $\boldsymbol{Y}_{\text{nc}}$ and $\boldsymbol{Y}_{\text{vc}}$ individually before combining them to obtain the full matrix $\boldsymbol{Y}$ as in (1). The result of this partial ordering strategy is that the upper left part of the factors $\boldsymbol{L}, \boldsymbol{U}$ becomes adequately sparse but the lower right part of the factors contains a very large number of fill-ins. This problem that slows down the algorithm considerably is illustrated in Fig. 2, and a theoretical explanation of the excessive fill-in is given below.

5.2 Sparse factorisation and dense Schur complement

The Schur complement (4) can also be obtained by successively *eliminating* nodes from the system and creating reduced admittance matrices. If the kth node is to be eliminated from an N node network as illustrated in Fig. 3, the new $(N-1) \times (N-1)$ admittance matrix is given by the formula

$$\boldsymbol{Y}^{\text{new}}_{(i,j)} = \boldsymbol{Y}_{(i,j)} - \frac{\boldsymbol{Y}_{(i,k)}\boldsymbol{Y}_{(k,j)}}{\boldsymbol{Y}_{(k,k)}}, \tag{5}$$

for $i, j \neq k$. The Schur complement S of the voltage controlled part is the matrix resulting from eliminating all *nc*s, see, e.g. [18].

Successive node elimination using the update formula (5) produces an equivalent network matrix that has fewer nodes; however, branches are added to the network, and the resulting network is therefore potentially less sparse. In the completely reduced network consisting of all *vc* nodes, the majority of nodes will be connected by branches. The Schur complement S is therefore a dense matrix.

In [28, 29], it is observed that if a matrix with the block structure in (1) is $\boldsymbol{LU}$-factored, the product of the lower right blocks $\boldsymbol{L}_{\text{vc}}, \boldsymbol{U}_{\text{vc}}$ of the factors $\boldsymbol{L}, \boldsymbol{U}$ corresponding to the *vc* part of the network gives the Schur complement of the voltage controlled part of the network directly, i.e. $S = \boldsymbol{L}_{\text{vc}}\boldsymbol{U}_{\text{vc}}$. This provides a way to compute and factor S but it also tells us why the large number of fill-ins are observed when computing Thevenin impedances with the method of [24] where the full matrix $\boldsymbol{Y}$ is factorised: because S is dense, the factors $\boldsymbol{L}_{\text{vc}}$ and $\boldsymbol{U}_{\text{vc}}$ will in general not be sparse. The product of sparse matrices can be dense, however, if, e.g. the node degree of the networks represented by the factors is limited, the product will be sparse. $\boldsymbol{L}_{\text{vc}}, \boldsymbol{U}_{\text{vc}}$ are precisely the lower right blocks of the factors $\boldsymbol{L}, \boldsymbol{U}$ where the excessive number of fill-ins occur. Indeed, any fixed bound on the maximum node degree in both $\boldsymbol{L}_{\text{vc}}$ and $\boldsymbol{U}_{\text{vc}}$ would imply that the number of non-zeros in S would grow linearly with the number of *vc*s, i.e. M. Since S is dense, the number of non-zeros grow quadratically, $nnz(S) \simeq M^2$, and no such bound can therefore exist.

6 Reduce–factor–solve Thevenin impedances

The factorisation of the dense Schur complement completely dominates the runtime when computing Thevenin impedances using the above outlined method. Therefore, a great speed-up of the computation can be achieved if the factorisation of the full admittance matrix $\boldsymbol{Y}$ and the excessive fill-in in the *nc*-part of the factors can be avoided. The *reduce–factor–solve* approach achieves exactly that.

The name of the method relates to its composition into three individual steps. The derivation below couples a variant of (2) with the structure of left-looking $\boldsymbol{LU}$-factorisation algorithms. For each *vc* node k, let $\boldsymbol{Y}_k$ denote the $(N - M + 1) \times (N - M + 1)$ admittance matrix of the system with all *vc* nodes but node k removed. A close variant of (2) for computing $Z^{-1}_{\text{th},k}$ uses the matrices $\boldsymbol{Y}_k$ instead of $\boldsymbol{Y}$. A factorisation $\boldsymbol{Y}_k = \boldsymbol{L}_k\boldsymbol{U}_k$ gives the relation

$$Z_{\mathrm{th},k}^{-1} = \boldsymbol{Y}_{(k,k)} - \hat{\boldsymbol{L}}_{k,(N-M+1)\cdot}\hat{\boldsymbol{U}}_{k,\cdot(N-M+1)} \tag{6}$$

where the notation in the rightmost term denotes the inner product between entries $1, \dots, N-M$ of the last row of $\boldsymbol{L}_k$ and of the rightmost column of $\boldsymbol{U}_k$. The advantage of using this formula is that the row $\hat{\boldsymbol{L}}_{k,(N-M+1)\cdot}$ and the column $\hat{\boldsymbol{U}}_{k,\cdot(N-M+1)}$ can be obtained from a factorisation $\boldsymbol{Y}_{\mathrm{nc}} = \boldsymbol{L}_{\mathrm{nc}}\boldsymbol{U}_{\mathrm{nc}}$ of the *nc*-part of $\boldsymbol{Y}$ only.

Consider now iterations of left-looking ***LU***-factorisation algorithms [25]. With this class of algorithms, the $N-M+1$ columns in a factorisation $\boldsymbol{Y}_k = \boldsymbol{L}_k\boldsymbol{U}_k$ are computed iteratively from left to right, i.e. starting with column 1 and ending with column $N-M+1$. At each step j, the upper left $(j-1)$-block of $\boldsymbol{L}_k$ is used to compute the first $j-1$ entries of the jth column of $\boldsymbol{U}_k$. In particular, computation of $N-M$ entries of the rightmost column uses only the upper left $(N-M)$-block of $\boldsymbol{L}_k$, i.e. the block representing the *nc*s. Writing this last step of the algorithm explicitly, the first $N-M$ entries of column $N-M+1$ of $\boldsymbol{U}_k$ satisfy the equation

$$\boldsymbol{L}_{\mathrm{nc}}\hat{\boldsymbol{U}}_{k,\cdot(N-M+1)} = \hat{\boldsymbol{Y}}_{\mathrm{link},\cdot k} \tag{7}$$

where $\hat{\boldsymbol{Y}}_{\mathrm{link},\cdot k}$ denotes the first $N-M$ entries of the column $\boldsymbol{Y}_{\mathrm{link},\cdot,k}$. The column vector $\hat{\boldsymbol{U}}_{k,\cdot(N-M+1)}$ is therefore computed with a triangular forward solve using the factorisation of $\boldsymbol{Y}_{\mathrm{nc}}$ only. Similarly, the first $N-M$ entries of row $N-M+1$ of $\boldsymbol{L}_k$ can be obtained from the equation

$$\boldsymbol{U}_{\mathrm{nc}}^T\hat{\boldsymbol{L}}_{k,(N-M+1)\cdot}^T = \hat{\boldsymbol{Y}}_{\mathrm{link},k\cdot}^T \tag{8}$$

again using only the factorisation of $\boldsymbol{Y}_{\mathrm{nc}}$. Thus, using (6), we get $\boldsymbol{Z}_{\mathrm{th},k}$ from two forward solutions using the factorisation of $\boldsymbol{Y}_{\mathrm{nc}}$.

With the above computations, all matrices and operations involved are sparse and the fill-in producing factorisation of the full admittance matrix $\boldsymbol{Y}$ is avoided. In addition, the triangular matrices used for the forward solves are not dependent on k, and the factorisation of $\boldsymbol{Y}_{\mathrm{nc}}$ must therefore be done only once. Due to the sparsity, the forward solves are each computationally lightweight, and they can in addition be computed completely in parallel. In the sequel, the factorisation of $\boldsymbol{Y}_{\mathrm{nc}}$ is denoted the *factorisation step* and the forward solutions (7), (8) combined for the *forward solve step*. Though Section 9 will show that the forward solve step can dominate the runtime, the completely parallel nature of the loop over all *vc*s makes speeding up this step straight forward by splitting the computation of several compute cores. In contrast, the factorisation step is hard to parallelize and therefore in reality the limiting factor of the algorithm. This step is analysed below.

6.1 Node elimination and factorisation speed

The factorisation step of Algorithm 1 consist of the ***LU***-factorisation of $\boldsymbol{Y}_{\mathrm{nc}}$. We use the KLU solver [21] that is particularly optimised for matrices with sparsity structure equivalent to power network matrices, and it is therefore inherently difficult to improve the factorisation speed. Nevertheless, it turns out that the execution time of the factorisation step can be reduced by using that factorisation of the entire submatrix $\boldsymbol{Y}_{\mathrm{nc}}$ is not required in order to evaluate (6). Instead, node elimination prior to factorisation can be performed to produce an equivalent but smaller matrix. This part of the reduce–factor–solve algorithm is denoted the *reduction* step.

```
Algorithm 1: Reduce-factor-solve algorithm
$Y_{nc}$ ← Fast node elimination $Y_{nc}$ (reduction)
$L_{nc}, U_{nc}$ ← factorisation of $Y_{nc}$ (factorisation)
for $k = N - M + 1 \rightarrow N$ do for each vc in parallel
    $\hat{U}_{k,\cdot(N-M+1)} \leftarrow \mathrm{solve}(L_{nc}, \hat{Y}_{\mathrm{link},\cdot k})$ (forward solve)
    $\hat{L}^T_{k,(N-M+1)\cdot} \leftarrow \mathrm{solve}(U^T_{nc}, \hat{Y}^T_{\mathrm{link},k\cdot})$
end for
for $k = N - M + 1 \rightarrow N$ do For Thevenin impedances
    $S_{(k-N+M,k-N+M)} \leftarrow$     $Y_{(k,k)} - \hat{L}_{k,(N-M+1)\cdot}\hat{U}_{k,\cdot(N-M+1)}$
  $Z_{\mathrm{th},k} \leftarrow S^{-1}_{(k-N+M,k-N+M)}$ Thevenin imp. node k end for
for $k, l = N - M + 1 \rightarrow N$ do For network reduction
    $S_{(k-N+M,l-N+M)} \leftarrow$(GPU)
            $Y_{(k,l)} - \hat{L}_{k,(N-M+1)\cdot}\hat{U}_{l,\cdot(N-M+1)}$ end for
```

Node elimination in the reduction-step can reduce the execution time of the factorisation step but careful consideration must be made with respect to the amount of fill–in generated by the elimination. Due to the efficiency of KLU, the reduction algorithm can be quite relaxed in removing only a relatively limited number of nodes. This is done with a simple fill-in reducing strategy: The algorithm scans through the *nc* nodes removing a node only if it is connected to <4 other *nc* nodes and if the fill introduced in the link matrix $\boldsymbol{Y}_{\mathrm{link}}$ is limited. Since removing nodes of degree 3 or less does not introduce fill-ins, this strategy ensures that the number of non-zeros in $\boldsymbol{Y}_{\mathrm{nc}}$ does not increase during the process, confer Fig. 3. The number of non-zeros in $\boldsymbol{Y}_{\mathrm{link}}$ will in general increase but the number of added fill-ins is controlled by a fixed limit. Additional methods for fill-in limiting eliminating can be found in the literature on large resistor networks, e.g. [30].

It will be shown in Section 9 that node elimination reduces the computational effort of the factorisation step by a factor of 2–3. Please note that this is a reduction of serial part of the algorithm that could not be obtained simply by adding more compute cores running in parallel.

The reduce–factor–solve algorithm is listed in Algorithm 1. Implementation details and pivoting issues are discussed in Section 8.

7 Fast network reduction

As the Thevenin impedances comprise the element-wise inverses of the diagonal of the Schur complement, the reduce–factor–solve approach can be regarded a fast algorithm for computing the diagonal of the Schur complement. Since the entire Schur complement is needed in transient stability applications, e.g. for computing the energy in the Lyaponov direct method and for identifying critical machines with the equal area criterion, the reduce–factor–solve approach will here be generalised to computing the entire Schur complement of the voltage controlled part of the network. As the Schur complement corresponds to node elimination, this operation is equivalent to eliminating all *nc*s from the network. The focus here is on reducing the execution time needed to perform this operation.

The Schur complement (4) can be obtained by computing the entire matrix instead of just the diagonal elements, i.e. by extending (6) to compute

$$\boldsymbol{Y}_{(k,l)} - \hat{\boldsymbol{L}}_{k,(N-M+1)\cdot}\hat{\boldsymbol{U}}_{l,\cdot(N-M+1)}$$

for all pairs $l, k = N-M+1, \dots, N$. This requires evaluation of M^2 inner products between sparse vectors instead of the M inner products needed for the Thevenin impedance computation. However, no additional evaluations in the forward solve step are needed. The performance of the algorithm is therefore dependent on the speed of evaluating the inner products.

Network reduction is often viewed as an iterative process of node elimination, and successive node elimination can be considered the most straight forward algorithm for network reduction. In order to eliminate all *nc*s, the update formula (5) must be applied $N-M$ times. Since each update step potentially updates the entire remaining matrix, this approach has complexity $O((N-M)N^2)$. For the first elimination steps, the matrix is very sparse and each update is in practice much faster than than the worst case $O(N^2)$ bound. As more nodes are eliminated, the number of added branches may grow quickly and reduce the sparsity of the matrix. For the last nodes to be eliminated, the matrix becomes dense. This fact limits the performance.

In contrast to the iterative process of node elimination is evaluating the Schur complement directly from (4) using matrix algebra. The inversion $\boldsymbol{Y}_{\mathrm{nc}}^{-1}$ will then in practice be carried out using an $\boldsymbol{LU}$-factorisation of the *nc*-part of the network. This approach is therefore equivalent to running Algorithm 1 with computations of all M^2 sparse inner products but without the reduction step.

With the terminology used in the $\boldsymbol{LU}$-factorisation literature, successive node elimination corresponds to a partial factorisation of the matrix using a *frontal* factorisation algorithm with $M - N$ rank-1 update steps. Similarly, direct evaluation of (4) corresponds to a frontal factorisation with one step and a sparse factorisation of the frontal matrix. The reduce–factor–solve approach can therefore be viewed as a hybrid between these algorithms that performs rank-1 updates until the sparsity starts decreasing followed by sparse $\boldsymbol{LU}$-factorisation for the remaining *vc* nodes.

8 Implementation and parallelisation

The execution time of Algorithm 1 is highly dependent on the *parallelisation* over multiple CPU cores or on highly parallel GPUs. This section provides details on the possibilities for parallelisation and the choices made in the actual implementation. A prototype implementation of the algorithm for computing both Thevenin impedances and for performing network reduction is available online [see https://bitbucket.org/stefansommer/networkred].

8.1 Reduction step

In the reduction step, *nc*-nodes that are connected to three or fewer additional *nc*-nodes are successively removed. The elimination of one node must be finished before its neighboring nodes can be removed. Performing node elimination in parallel will therefore require a dependency graph between the parallel parts and synchronisation between parallel threads. As each individual node elimination can be performed very fast, the overhead of such an approach may remove any gain from the parallel execution.

For each single node elimination, the algorithm updates the entries of the matrix corresponding to the branches and nodes of the connecting neighbors. The total number of connected nodes, i.e. in both the *nc*- and *vc*-part of the network, is in practice sufficiently large that some fine-grained parallelism can be exploited in this operation. In order to avoid overhead in synchronising threads, SIMD (single instruction, multiple data) vector instructions are used to perform the updates for four branches at a time. This produces a two to three times speed-up of the node-elimination process.

In each successive node elimination, the update formula (5) requires division by the diagonal element of the node to be eliminated, i.e. $\boldsymbol{Y}_{(k,k)}$ when eliminating the *k*th node. For factorisation algorithms in general, attention must be payed to the numerical stability when the diagonal value is small compared to the off-diagonal element. The division can be avoided by pivoting rows or columns in the matrix [25].

Admittance matrices representing power grids may not be diagonally dominant, and diagonal entries with small absolute value can occur e.g. due to the addition of shunts. Fortunately, small diagonal entries can be handled elegantly in the reduction step without introducing complicated pivot schemes: If a node is encountered with low absolute value, the elimination can just skip the node. The subsequent factorisation performed by the $\boldsymbol{LU}$-factorisation algorithm then takes care of the pivoting. Thus, numerical stability can be ensured by merely adding a runtime-check for small diagonal entries.

8.2 Forward solve step

Both the forward solutions (7), (8) and the following computation of inner products are completely parallel operations. Without parallelisation, these steps will dominate execution time. In particular, computing the M^2 inner products for network reduction is time consuming.

The 2*M* solutions involved in the forward solve steps can be run in parallel. Each individual solution is a serial operation and can in general be computed very fast using a sparse forward solution. In addition, if no pivoting occurs, the evaluation tree can be precomputed for each row and column reducing the execution time for the numerical solution. The whole process can be efficiently executed on e.g. a multi-core CPU using several threads. For the Thevenin impedance, the subsequent M inner products can in addition efficiently be calculated using the same threads. The algorithms used for each forward solution and sparse inner product is described in [25].

The quadratic scaling of the M^2 inner products for network reduction forces a different approach for the network reduction algorithm. The limited number of CPU cores on present machines makes the effect of parallelisation limited: in Section 9, the algorithm will be evaluated on power systems that allow evaluation of $>10^6$ inner products in parallel but a standard machine usually have <12 cores. This difference suggests using massively parallel execution units such as GPUs.

A typical GPU allows a much larger number of execution units to work in parallel than a CPU, and the large number of cores can significantly speed-up the evaluation of the M^2 inner products. Usually, the work flow when employing GPUs consist of a transfer from the main computer memory to the GPU memory, execution of the GPU program, and transfer of the result from GPU memory to the main computer memory. As the inner product computation can be expressed as a multiplication of two sparse matrices, a general sparse matrix multiplication kernel can be used to do the actual computation on the GPU card. For this task, the cuSPARSE library is used [https://developer.nvidia.com/cusparse].

The actual GPU computation is relatively fast compared to the memory transfer operations. This is in particular amplified by the fact that cuSPARSE requires one of the two matrices to be multiplied to be structured as a dense matrix. Since matrices of dimension $M \times N - M$ and $N - M \times M$ are multiplied to obtain a matrix of dimension $M \times M$, the transfer memory time is dependent on the size of *N*. Conveniently, the reduction step of the reduce–factor–solve algorithm removes a large part of the *nc* nodes before the inner product calculation. This produces a sufficient reduction in memory transfer time. See, e.g. [31, 32] for additional ongoing research on the data structures used for sparse GPU computations

9 Experiments

In the first set of experiments, the speed-up provided by the reduce–factor–solve Thevenin impedance algorithm, its absolute runtime, and its validity is evaluated. In particular, the experiments will show that the Thevenin impedance of all generators for power systems of considerable sizes can be established in <3 ms. In addition, the runtime of the serial and parallel parts of the algorithm will be explored in order to evaluate the achieved overall efficiency, and a great reduction in size and number of non-zeros for the matrix to be factored will be observed.

In the second set of experiments, network reduction with the reduce–factor–solve approach is considered and the effect of parallelisation and GPU computation is investigated. In particular, it will be shown how the reduction step reduces the time spent on host GPU memory transfer.

The algorithms will be compared to the existing method described in [24]. In addition, multiple methods for computing the Schur complement will be explored. The validity of the algorithms is ensured by measuring the differences in the output of the methods. For all experiments, it is observed that the results are equal up to numerical precision showing that the new algorithms produce correct results.

The experiments are performed on admittance matrices generated from test systems included in the PSS® E-30.0 and MATPOWER [33] network simulation packages [see http://www.pserc.cornell.edu/matpower/docs/ref/matpower5.0/case*.html and https://bitbucket.org/stefansommer/networkred/src/master/data/. Only test systems with sufficient size to ensure non-negligible runtime are included in the evaluation]. The test systems include the US west-coast (1648 buses, 2602 branches) and US east-coast (7917 buses, 13,014 branches) power grids along with 6 additional systems ranging from 2383 to 3120 buses.

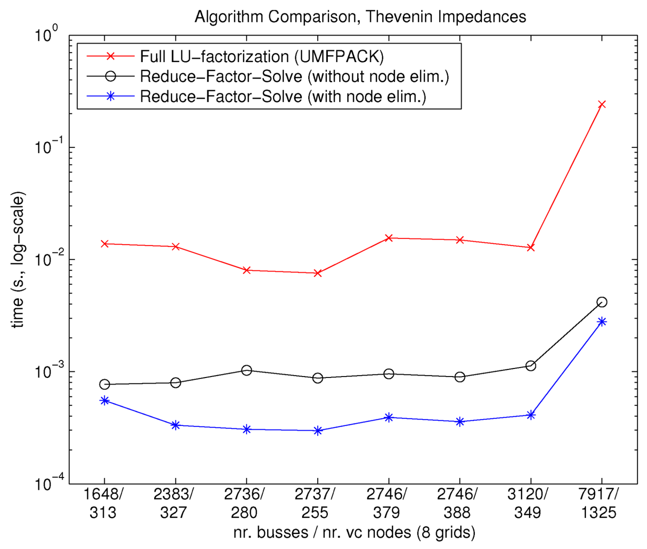

Fig. 4 *Computation time for determining Thevenin impedances when factorising the full admittance matrix ([24], red), the reduce–factor–solve algorithm without node elimination (black), and the reduce–factor–solve algorithm with node elimination (blue). Evaluation performed on 8 power grids ranging from 1648 buses to 7917 buses with between 313 and 1325 vcs. Note the logarithmic scale on the time axis. For the largest system, the new method is roughly 80 times faster than the previous approach*

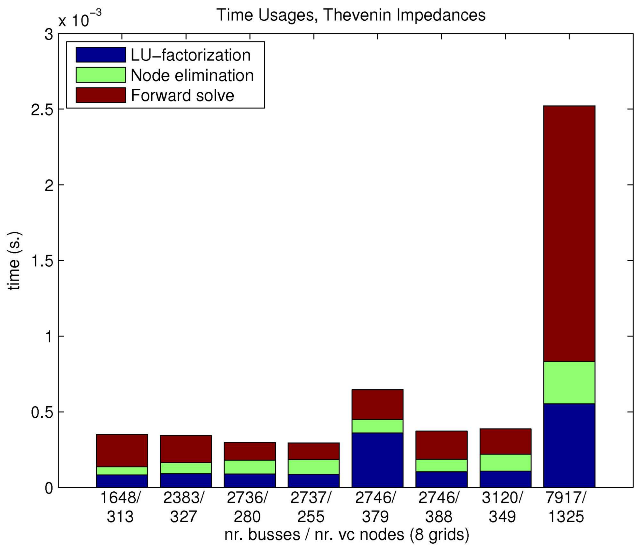

Fig. 5 *Timings for the three different parts of the reduce–factor–solve algorithm: factorisation (blue), node elimination (green), and forward solve (red). The forward solve step parallelizes completely and the runtime can thus be reduced by employing more computational cores*

The runtime is tested on a 3.2 GHz Intel Core i7 hexa-core CPU. In accordance with [24], UMFPACK [21] is used for factoring the full admittance matrix with the reference method, and KLU [21] is used for the factorisation step of Algorithm 1. Pivoting is disabled for all factorisation methods. The main loop of the algorithm is parallelised over the CPU cores using threads, and the node elimination step uses SIMD instructions to exploit fine-grained parallelism. The GPU computations are performed on NVIDIA Titan Xp GPUs with 12 GB of memory and 3840 CUDA cores per card.

9.1 Thevenin impedance computation

Fig. 4 shows for each test system the runtime of the Thevenin impedance algorithm employing $\boldsymbol{LU}$-factorisation of the full admittance matrix, the runtime of the reduce–factor–solve algorithm without node elimination, and the reduce–factor–solve algorithm with node elimination prior to the factorisation. For all three approaches, the runtime of the initial preparation step is left out of the measurements because this step only need to be done once for each network. The timings are performed just on the computational parts leaving out the time used for initial copying of data, and the obtained timings are averaged over a large number of runs. Note the logarithmic scale on the vertical axis and the achieved ∼80 times speed-up on the largest system with the reduce–factor–solve algorithm compared to the previous method.

In Fig. 5, the runtime of the three different steps in the reduce–factor–solve algorithm is plotted: reduction, factorisation, and forward solve. It can be seen that a relatively large portion of the computational effort is spend on the forward solve. It is important to relate this to the fact that the forward solve step is completely parallelizable. For the results here, all six cores of the test machine are used. If a reduction in runtime is needed, a machine with more cores will allow the runtime of the forward solve step to be driven down below the runtime used for the reduction and factorisation.

Because the forward solve step can be parallelised, the serial parts of the algorithm are in reality the true bottlenecks. In Fig. 6, the runtime of the serial parts are plotted in order to evaluate the benefits of the node elimination step. Employing node elimination results in a two to three times speed-up for this part of the algorithm: for the largest test system, the factorisation time without node elimination is 2.4 ms. With node elimination, elimination and factorisation takes 0.8 ms combined. In terms of sparsity, for the largest test system, the reduce–factor–solve algorithm reduces the dimension of the matrix to be factorised from 7917×7917 (the full admittance matrix) to 960×960 (the node eliminated non-controlled part of the admittance matrix). At the same time, the number of non-zeros in the factors is reduced from 1,549,093 to 10,174.

9.2 Network reduction

Progressing to full network reduction, Fig. 7 shows the execution time when reducing the entire *nc* part of the network. As previously discussed, the reduce–factor–solve algorithm can be seen as a hybrid between node elimination and direct evaluation of the matrix (4). Therefore, these two approaches and the reduce–factor–solve Algorithm 1 are compared. The tests are performed with and without GPU acceleration using one GPU.

The differences between the methods are the potential for parallelisation, and, for the GPU code, the time spent on memory transfer. Node elimination uses parallelisation exclusively inside each elimination step, and therefore consistently performs worse than the remaining methods that allows more parallelisation. For the largest system, the reduce–factor–solve algorithm with GPU acceleration reduces the execution time an order of magnitude compared to node elimination.

Comparing the GPU accelerated direct evaluation of (4) with the reduce–factor–solve approach, the speed–up is mainly a result of a reduction in memory transfer times. The node reduction step of the reduce–factor–solve approach decreases the number of *nc* nodes by roughly a factor 6 for the largest system resulting in an equivalent reduction in host-to-GPU memory transfer time. The GPU-to-host transfer of the computed result is not affected. Both approaches uses ∼6 ms for the actual computation but the memory transfer time is reduced from 44 to 8 ms with the reduce–factor–solve approach.

For all but the largest system, the total execution time for the reduction of the *nc* part of the network is below 10 ms For the largest network, the execution time is 14 ms. However, due to the parallelisation ability of the inner product calculation, using multiple GPU cards can further reduce the memory transfer and computation time. Using two GPUs, the execution time is approximately halved. Addition of more GPUs beyond two has little effect on the execution time as not all of the very large number of compute cores can be effectively utilised for systems of the size considered here. Larger systems may however benefit from computations using >2 GPUs.

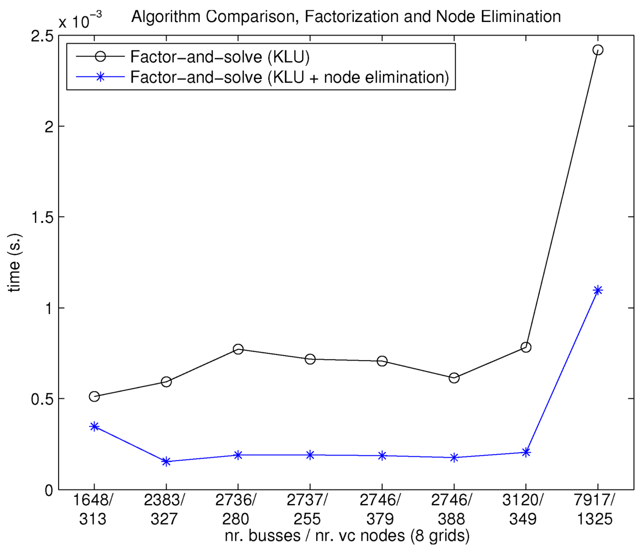

Fig. 6 *Runtime for the reduce–factor–solve algorithm excluding the forward solve step without node-elimination (black) and with node-elimination (blue). Node elimination results in a speed-up of a factor 2–3 for this part of the algorithm*

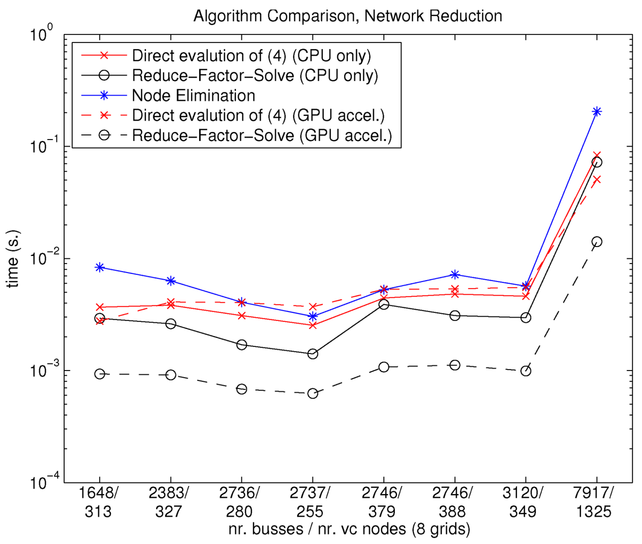

Fig. 7 *Computation time for network reduction of the nc nodes with direct evaluation of (4) using KLU for the factorisation and with/without GPU acceleration (red), the reduce–factor–solve algorithm with/without GPU acceleration (black), and node elimination only (blue). The CPU execution times are roughly equivalent for all but the largest system. The reduce–factor–solve approach with GPU acceleration consistently improves the execution time by approximately a factor of 5. For the largest system, the GPU accelerated reduce–factor–solve approach is six times faster than evaluating (4) directly*

10 Future work

The algorithms discussed in this paper concerns some of the steps involved in common transient stability methods. Online application of these methods will require fast computation of all the involved operations. In particular, a number of algorithms including the Lyaponov method and EAC solves linear systems involving the reduced admittance matrix. In this paper, it is shown how to compute the reduced matrix fast, e.g. for evaluating the Lyaponov energy. However, because the reduced matrix is dense, the network sparsity cannot be exploited when solving the mentioned linear system. The speed-up of the reduce–factor–solve approach lies precisely in the ability to avoid dense sub-matrices, and we are therefore working on applying similar approaches to reducing the execution time of the remaining operations needed for the transient stability methods.

11 Conclusion

Real-time calculation of different properties of the grid operation state is necessary for online stability and security assessment. In particular, calculating Thevenin impedances and performing network reduction is important for several suggested approaches to stability and security assessment. The paper introduces the reduce–factor–solve approach that allows computation of both Thevenin impedances and fast network reduction, and it describes the theoretical foundation and implementation details for the algorithm.

The performance and validity of the approach is tested on several power systems. Comparison with previous approaches shows ∼80 times speed-up for the largest power system for calculation of Thevenin impedances and 5 times speed-up when performing network reduction using GPU acceleration. Consequently, neither Thevenin impedance computation or network reduction constitute bottlenecks for real-time stability and security assessment.

12 Acknowledgments

This work is part of the Security Assesment of Renewable Power Systems (SARP) project with support from ForskEL- Energinet.dk.

13 References

[1] Li, F., Qiao, W., Sun, H., *et al.*: 'Smart transmission grid: vision and framework', *IEEE Trans. Smart Grid*, 2010, **1**, (2), pp. 168–177

[2] Sommer, S., Johannsson, H.: 'Real-time Thevenin impedance computation'. 2013 IEEE Power & Energy Society (PES) Innovative Smart Grid Technologies (ISGT), Washington, DC, USA, February 2013, pp. 1–6

[3] Jóhannsson, H., Nielsen, A.H., Østergaard, J.: 'Wide-area assessment of aperiodic small signal rotor angle stability in real-time', *IEEE Trans. Power Syst.*, 2013, **28**, (4), pp. 4545–4557

[4] Jóhannsson, H., Østergaard, J., Nielsen, A.H.: 'Identification of critical transmission limits in injection impedance plane', *Int. J. Electr. Power Energy Syst.*, 2012, **43**, (1), pp. 433–443

[5] Corsi, S., Taranto, G.: 'A real-time voltage instability identification algorithm based on local phasor measurements', *IEEE Trans. Power Syst.*, 2008, **23**, (3), pp. 1271–1279

[6] Smon, I., Verbic, G., Gubina, F.: 'Local voltage-stability index using Tellegen's theorem'. IEEE Power & Energy Society (PES) General Meeting, Tampa, FL, USA, June 2007

[7] Vu, K., Begovic, M., Novosel, D., *et al.*: 'Use of local measurements to estimate voltage-stability margin', *IEEE Trans. Power Syst.*, 1999, **14**, (3), pp. 1029–1035

[8] Warland, L., Holen, A.: 'Estimation of distance to voltage collapse: testing an algorithm based on local measurements'. Power Systems Computation Conf. (PSCC), Seville, Spain, June 2002

[9] Perez, A., Jóhannsson, H., Østergaard, J.: 'Wind farms generation limits and its impact in real-time voltage stability assessment'. IEEE PowerTech 2015, Eindovhen, June 2015

[10] Perez, A., Jóhannsson, H., Østergaard, J.: 'Improved method for considering PMU's uncertainty and its effect on real-time stability assessment methods based on Thevenin equivalent'. IEEE PowerTech 2015, Eindovhen, June 2015

[11] Perez, A., Jóhannsson, H., Cutsem, T.V., *et al.*: 'Improved Thevenin equivalent methods for real-time voltage stability assessment'. 2016 IEEE Int. Energy Conf., Leuven, Belgium, April 2016

[12] Møller, J., Jóhannsson, H., Østergaard, J.: 'Fast computation of steady state nodal voltages in power system contingency studies'. 2014 IEEE Power Quality and Reliability Conf., Tallin, Estonia, June 2014

[13] Møller, J.G., Jóhannsson, H., Østergaard, J.: 'Super-positioning of voltage sources for fast assessment of wide-area Thevenin equivalents', *IEEE Trans. Smart Grid*, 2017, **8**, (3), pp. 1488–1493

[14] Møller, J., Jóhannsson, H., Østergaard, J.: 'Contingency assessment with detection of aperiodic small-signal instability'. 2015 IEEE Power & Energy Society (PES) General Meeting, Denver, Colarado, July 2015

[15] Gless, G.: 'Direct method of Liapunov applied to transient power system stability', *IEEE Trans. Power Appar. Syst.*, 1966, **PAS-85**, (2), pp. 159–168

[16] Weckesser, T., Jóhannsson, H., Sommer, S., *et al.*: 'Investigation of the adaptability of transient stability assessment methods to real-time operation'. IEEE Power & Energy Society (PES) Innovative Smart Grid Technologies (ISGT), Europe, Berlin, 2012

[17] Pavella, M., Ernst, D., Ruiz-Vega, D.: '*Transient stability of power systems: a unified approach to assessment and control*' (Kluwer Academic Publishers, Norwell, MA, USA, 2000, 1st edn.)

[18] Dorfler, F., Bullo, F.: 'Kron reduction of graphs with applications to electrical networks', *IEEE Trans. Circuits Syst.*, 2011, **60**, (1), pp. 150–163

[19] Belkacemi, M., Harid, N.: 'Fast reduction and modification of power system sparse matrices', *Electr. Power Compon. Syst.*, 2004, **32**, (4), pp. 367–373

[20] Abu-Elnaga, M., El-Kady, M., Findlay, R.: 'Sparse formulation of the transient energy function method for applications to large-scale power systems', *IEEE Trans. Power Syst.*, 1988, **3**, (4), pp. 1648–1654

[21] Davis, T.A., Palamadai Natarajan, E.: 'Algorithm 907: KLU, a direct sparse solver for circuit simulation problems', *ACM Trans. Math. Softw.*, 2010, **37**, (3), pp. 36:1–36:17

[22] Ilic'-Spong, M., Crow, M.L., Pai, M.A.: 'Transient stability simulation by waveform relaxation methods', *IEEE Trans. Power Syst.*, 1987, **2**, (4), pp. 943–949

[23] Pruvost, F., Cadeau, T., Laurent-Gengoux, P., *et al.*: 'Numerical accelerations for power systems transient stability simulations'. 17th Power System Computation Conf., Stockholm, Sweden, August 2011

[24] Jóhannsson, H.: 'Development of early warning methods for electric power systems'. PhD dissertation, Technical Univ. of Denmark, 2011

[25] Davis, T.A.: '*Direct methods for sparse linear systems*' (SIAM, Philadelphia, PA, USA, 2006)

[26] Zhang, F. (Ed.): '*The Schur complement and its applications, ser. Numerical methods and algorithms*', vol. 4 (Springer, New York, NY, USA, 2005)

[27] Amestoy, P.R., Davis, T.A., Duff, I.S.: 'Algorithm 837: AMD, an approximate minimum degree ordering algorithm', *ACM Trans. Math. Softw.*, 2004, **30**, (3), pp. 381–388

[28] Saad, Y.: '*Iterative methods for sparse linear systems*' (SIAM, Philadelphia, PA, USA, 2003)

[29] Saad, Y., Sosonkina, M.: 'Distributed Schur complement techniques for general sparse linear systems', *SIAM J. Sci. Comput.*, 1997, **21**, pp. 1337–1356

[30] Ye, Z., Vasilyev, D., Zhu, Z., *et al.*: 'Sparse implicit projection (SIP) for reduction of general many-terminal networks'. IEEE/ACM Int. Conf. on Computer-Aided Design, San Jose, CA, USA, November 2008

[31] Matam, K., Kothapalli, K.: 'Accelerating sparse matrix vector multiplication in iterative methods using GPU'. Int. Conf. on Parallel Processing (ICPP), Taipei City, Taiwan, September 2011

[32] Buluç, A., Gilbert, J.R.: 'Parallel sparse matrix–matrix multiplication and indexing: implementation and experiments', *SIAM J. Sci. Comput.*, 2012, **34**, (4), pp. 170–191

[33] Zimmerman, R., Murillo-Sanchez, C., Thomas, R.: 'MATPOWER: steady-state operations, planning, and analysis tools for power systems research and education', *Trans. Power Syst.*, 2011, **26**, pp. 12–19