GPU-Based Criticality Analysis Applied to Power System State Estimation

Ayres Nishio da Silva Junior(✉), Esteban W. G. Clua, Milton B. Do Coutto Filho, and Julio C. Stacchini de Souza

Universidade Federal Fluminense, Niteroi, RJ, Brazil
ayresnishio@gmail.com

Abstract. State Estimation (SE) is one of the main tools in the operation of power systems. Its primary role is to filter statistically small errors and to eliminate spurious measurements. SE requires an adequate number of measurements, varied, and strategically distributed in the power grid. Criticality analysis consists of identifying which combinations of measurements, forming tuples of different cardinalities, are essential for observing the power network as a whole. If a tuple of measurements becomes unavailable and, consequently, unobservable, the tuple is considered critical. The observability condition is verified by the factorization of the residual covariance matrix, which usually is a time-consuming computation task. Also, the search for criticalities is costly, being a combinatorial based problem. This paper proposes a parallel approach of the criticality analysis in the SE realm, through multi-threads execution on CPU and GPU environments. To date, no publication reporting the use of GPU for computing critical elements of the SE process is found in the specialized literature. Numerical results from simulations performed on the IEEE 14- and 30-bus test systems showed speed-ups close to 25×, when compared with parallel CPU architectures.

Keywords: High performance computing · Supercomputing · Critical analysis · Power system state estimation

1 Introduction

The Energy Management System (EMS) found in control centers encompasses computational tools to monitor, control, and optimize the operation of power networks. Among these tools, the State Estimation (SE) is responsible for processing a set of measurements to obtain the system operating state [1,12]. The SE process depends on the number, type, and distribution of measurements throughout the grid. The observability/criticality analysis reveals the vulnerability of the measuring system to feed SE adequately. This analysis aims to discover the strengths and weaknesses of measurement plans, allowing preventive

O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12251, pp. 121–133, 2020.
https://doi.org/10.1007/978-3-030-58808-3_10

actions against possible cyber-attacks, and offering different choices of measurement reinforcements. However, due to its combinatorial characteristics, criticality analysis is computationally very expensive.

The word critical is used in the paper to denote extremely important or essential elements of the measuring system devoted to SE. In this sense, the following definitions are stated:

- Critical Measurement (C_{meas}): when not available, it makes the complete system unobservable.
- Critical k-Tuple (C_k): the set of k measurements that, when simultaneously unavailable, leads the system to an unobservable state;

Besides the remarked unobservability cases, the presence of C_{ks} compromises the SE credibility as a function capable of providing a dataset free of gross errors [3]. When there are one or more C_{meas} in the available measurement set, it is impossible to detect any error in the value of these measurements coming from the SE residual analysis. On the other hand, if $(k-2)$ measurements with gross errors belong to a C_k, they can be detected/identified, but if $(k-1)$ or (k) spurious measurements are present in this C_k, they can only be detected.

The criticality analysis becomes computationally costly, particularly when the cardinality of critical tuples k increases, as the consequence of the factorial number of possible combinations to analyze.

This paper presents a parallel programming approach to speed up the process of searching for criticalities (of single measurements or when they are considered forming groups), in which diverse combinations of measurements are simultaneously analyzed. Based on the best knowledge existing so far, there is no publication concerning the use of GPU for identifying critical elements of the SE process is found in the specialized literature. The results (obtained in a multi-core GPU environment) of simulations performed on the IEEE 14- and 30-bus test systems validate the proposed approach.

This paper is organized as follows: Sect. 2 presents some previous works related to criticality analysis. Section 3 presents the residual covariance matrix and its respective application to solve the problem. Section 4 presents our proposed algorithm and it's adaptations to parallel computing. Section 5 shows the results of the implementations in CPU and GPU environments. Finally, Sect. 6 concludes this work.

2 Related Works

One of the first studies on criticality analysis and its relation with the detection/identification of gross measurement errors can be found in [6]. As the identification of all C_{ks} present in a measurement system is a difficult optimization problem [14], most of the published works were limited to study small cardinalities ($k \leq 3$). This paper adopts the method based on the residual covariance matrix to identify C_{ks} [3].

Criticality analysis is a useful tool to evaluate the robustness of measurement schemes [5,7], as well as to prevent possible cyber-attacks of false data injections, as described in [10,15]. In this type of attack, a malicious attacker intentionally adds gross errors to the available measurements. Based on previous knowledge of the measurement criticalities, it is possible to take preventive actions (e.g., to reinforce the metering plan), so that the effect of an attack can be mitigated.

Few studies have been conducted to address the general problem of searching for C_{ks}. Reference [2] tackles this hard combinatorial problem via the Branch-and-Bound (B&B) approach. B&B is a tree-based exploratory tool for finding implicitly exact solutions.

Parallel programming has been used to speed up the estimation filtering process [9,10]. B&B algorithms in GPUs [4] have also been implemented. The GPU approach to deal with criticality analysis in SE using the residual covariance matrix method can be considered the main contribution of the present paper.

3 The Residual Covariance Matrix Method

The entries of the residual covariance matrix (Ω) represent the degree of interaction between measurements, which can be explored in the criticality analysis [3]. Also, using the definition of a C_k previously introduced, the following properties are useful [6]:

1. The columns of matrix Ω associated with the measurements that form a C_k are linearly dependent.
2. A C_k does not contain a C_j, for $\forall k > j$.

Null elements in Ω indicate the presence of entirely uncorrelated measurements. Therefore, null rows/columns in Ω are associated with C_{meas}.

Now, consider $\tilde{\Omega}$ denoting a submatrix of Ω, composed of the columns/rows associated with a group of k measurements. If $\tilde{\Omega}$ has linearly dependent columns/rows, then the measurements associated with those form a C_k. Gauss elimination can be used to factorize $\tilde{\Omega}$ to check whether there are linearly dependent rows/columns in the matrix.

To gain insight into criticalities, for instance, consider the six-bus system represented in Fig. 1, measured by five power flows and four power injections, for which all C_{ks} present in the set of measurements will be identified.

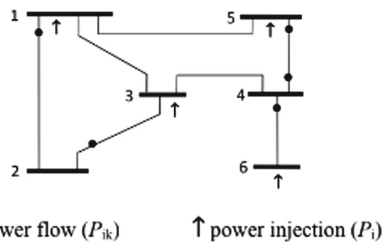

Fig. 1. Six bus system, with its Measurement set

Symmetric matrix Ω, obtained with the decoupled model for the adopted test system and usually adopted in the classical observability analysis [1] is presented in Table 1.

Table 1. Covariance matrix Ω, obtained from six bus system depicted in Fig. 1

Meas. #	1	2	3	4	5	6	7	8	9
	P_{1-2}	P_{2-3}	P_{4-5}	P_{4-6}	P_{5-4}	P_1	P_3	P_5	P_6
P_{1-2}	0.57	0.38	-0.07	0.00	0.07	-0.23	0.05	-0.19	0.00
P_{2-3}		0.53	0.16	0.00	-0.16	-0.02	0.17	0.15	0.00
P_{4-5}			0.66	0.00	0.34	0.13	0.10	0.23	0.00
P_{4-6}				0.50	0.00	0.00	0.00	0.00	0.50
P_{5-4}					0.66	-0.13	-0.10	-0.23	0.00
P_1						0.17	0.05	0.21	0.00
P_3							0.07	0.12	0.00
P_5								0.34	0.00
P_6									0.50

As one can observe, there is no null columns/rows in Ω; thus, the set of measurements is free from C_{meas}. Now, concerning, for instance, the tuples of cardinality two, a C_2 is identified, involving the measurements #4 and #9, i.e., branch $4-6$ power flow ($P_{4,6}$) and the power injection at bus 6 (P_6). The submatrix $\tilde{\Omega}_{4,9}$ is built with the entries related to the respective rows/columns:

$$(\tilde{\Omega}_{4,9}) = \begin{bmatrix} 0.50 & 0.50 \\ 0.50 & 0.50 \end{bmatrix} \tag{1}$$

$$(\tilde{\Omega}_{4,9}) = \begin{bmatrix} 0.50 & 0.50 \\ 0.00 & 0.00 \end{bmatrix} \tag{2}$$

For tuples of cardinality 3, for instance, rows #2, #7, and #8 are associated with measurements (P_{2-3}, P_3, P_5) that form a C_3. Submatrices $\Omega_{2,7,8}$ in (3) and its equivalent triangular form in (4), support this conclusion.

$$(\tilde{\Omega}_{2,7,8}) = \begin{bmatrix} 0.53 & 0.17 & 0.15 \\ 0.17 & 0.07 & 0.12 \\ 0.15 & 0.12 & 0.33 \end{bmatrix} \tag{3}$$

$$(\tilde{\Omega}_{2,7,8}) = \begin{bmatrix} 0.53 & 0.06 & 0.15 \\ 0.00 & -0.05 & 0.22 \\ 0.00 & 0.00 & 0.00 \end{bmatrix} \tag{4}$$

Together with (P_{2-3}, P_3, P_5) nine more C_3 are identified. They are: (P_{1-2}, P_{2-3}, P_1), (P_{1-2}, P_{2-3}, P_3), (P_{1-2}, P_{2-3}, P_5), (P_{1-2}, P_1, P_3), (P_{1-2}, P_1, P_5), (P_{1-2}, P_3, P_5), (P_{2-3}, P_1, P_3), (P_{2-3}, P_1, P_5), (P_1, P_3, P_5). Table 2 summarizes the results achieved for all C_{ks}. There is an upper limit (k_{lim}) for the cardinality of C_{ks}, which is dependent of the difference between m (number of available measurements) and n (number of state variables), given by: $k_{lim} = m - n + 1$. Thus, for the system under analysis, $k_{lim} = 9 - 6 + 1 = 4$.

Table 2. Number of visited combinations of measurements and C_{ks} identified in the 6-bus system depicted in Fig. 1

Cardinality k	No. of visited combinations of measurements	No. of C_{ks} identified
1	9	0
2	36	1
3	84	10
4	126	10
5	126	0
6	84	0
7	36	0
8	9	0
9	1	0
Total	511	21

4 Proposed Algorithm

The proposed algorithm receives as input Ω, m, and k_{max} and returns as output the solution set ($solSet$) containing all criticalities C_{ks} in the measurement set. The solution set contains binary vectors, in which the elements that correspond measurements removed from the set are represented by 1. Figure 2 depicts the C_{ks} that were found in the illustrative example of Sect. 4 stored in the $solSet$. As the number of C_k is previously unknown, $solSet$ was implemented as a stack.

P_{1-2}	P_{2-3}	P_{4-5}	P_{4-6}	P_{5-4}	P_1	P_3	P_5	P_6
0	0	0	1	0	0	0	0	1
0	1	0	0	0	0	1	1	0

Fig. 2. Examples of output representation for criticalities $\{P_{4-6}; P_6\}$ and $\{P_{2-3}; P_3; P_5\}$

The main algorithm adaptation to parallel programming considers the possibility to evaluate all possible k combinations of m measurements simultaneously.

Algorithm 1 shows that each cardinality is analyzed separately within a loop until the adopted k_{max} is reached. Furthermore, each loop consists of four steps, as shown in Algorithm 1. The following subsections detail each one of these steps.

The first step creates a combination matrix ($combMat$) ($\binom{k}{m} \times m$) to store all possible combinations of m measurements taken from k at a time. Steps 2 and 3 are the main focus of the present work, in which CPU and GPU parallel computing are adopted. Step 2 identifies combinations of measurements whose removal compromises the system observability. The third step checks whether or not the solution of the second step is indeed a C_k. Finally, step 4 updates the solution set ($solSet$).

Algorithm 1: Parallel Critcality Analysis

input : Ω, m, k_{max}
output: $ConjSol$
for $k \leftarrow 1$ **to** k_{max} **do**
  Step 1: Build Combinaton Matrix
  Step 2: Find out Possible C_ks
  Step 3: Confirm C_ks
  Step 4: Update Solution Set
end

4.1 *Step 1*: Combination Matrix

The construction of $combMat$ is an adaptation of the algorithm presented in [13]. The original algorithm allocates the measurement combinations in a stack so that they can be further successively accessed. However, to allow a parallel implementation, the combinations are represented by rows in a matrix data structure, enabling concurrent access to them.

4.2 *Step 2*: Searching for C_{ks}

At this step, the algorithm builds a submatrix $\tilde{\Omega}$ for each row in $combMat$. Then, the invertibility of each $\tilde{\Omega}$ is tested by using Gauss Elimination, identifying possible C_{ks}.

As can be seen in Algorithm 2, there is an auxiliary binary array $isCrit$ in which each element corresponds to a row in $combMat$. If the algorithm finds a possible C_k, it assigns 1 to its corresponding element in $isCrit$; otherwise, it assigns 0.

The multi-thread computing runs each loop in Algorithm 2 simultaneously. The array $isCrit$ enables threads to record which row of $combMat$ is a possible C_k concurrently. Distinctly, in GPU implementation, this step requires Ω and $combMat$ to be transferred to GPU global memory.

Algorithm 2: *Step 2:* Searching for C_{ks}

```
for i ← 0 to (k m) do
    build Ω̃(Ω, k, combMat);
    if Ω̃ is invertible then
        isCrit[i]=0;
    else
        isCrit[i]=1;
    end
end
```

4.3 *Step 3*: Confirmation

Considering C_j $(j < k)$ a criticality solution previously stored in $solSet$, the algorithm compares all C_j with all rows in $combMat$. If a row has its corresponding $isCrit = 1$ and contains any C_j, then the measurement combination found in step 2 is not a C_k. Consequently, the algorithm updates its related value in $isCrit$ to 0.

The algorithm uses an element-wise subtraction to verify if an array is a subset of another. For instance, considering $C_2 = \{1001\}$ as a critical set previously-stored in $solSet$ and $C_3 = \{1011\}$ as possible criticality found in step two. Subtracting each element of those arrays, $(C_3 - C_2) = \{0010\}$, then, since there was no element equal to -1 in such operation, one can conclude that $C_2 \subset C_3$. So C_3 is not considered critical, and its related element in $isCrit$ is set to 0.

Algorithm 3: Confirmation

```
for j in solSet do
    for i=0 : (k m) do
        if isCrit[i]==1 then
            if solSet[j] ⊂ matComb[i] then
                isCrit[i]==0;
            end
        end
    end
end
```

The multi-thread approach executes the inner loop in Algorithm 3 simultaneously. Also, for this implementation on GPU, $solSet$ must be initially transferred to the GPU global memory.

4.4 *Step 4*: Solution Set

In this last step, all the rows of $combMat$ with the corresponding $isCrit = 1$ are added to $solSet$, as shown in Algorithm 4.

Algorithm 4: Solution Set

```
for i =0 : (k m) do
    if isCrit[i]==1 then
        add(combs[i], solSet)
    end
end
```

5 Results

This section presents the comparative results between the sequential steps two and three and its parallel implementations in multi-core CPU and GPU environments. For GPU computing, such comparison also considered the data management between host and device.

Simulations carried out on a computer with 8 GB RAM, Intel Core i5-9300H processor, and NVIDIA GeForce GTX 1650 graphics card. The original single-threaded algorithm was implemented in C++. Furthermore, the CPU multi-threading was performed by OpenMP, using 8 threads and GPU computing in NVIDIA CUDA, using 512 threads in each kernel call.

From the performed simulations it could be noted that the use of parallelism did not prove to be advantageous in systems with a few measurements and for small values of k, which resulted in almost any combinations to be analyzed. Therefore, results for $k > 6$ using the IEEE 14-bus and IEEE 30-bus test systems will be presented.

Also, some values of k resulted in a large number of combinations that could not be stored at once in the combination matrix. Then, the k-value analysis (with more than 2^{20} combinations) was partitioned.

5.1 IEEE 14-Bus System

The IEEE 14-bus test case adopts the measuring system presented in [3], in which there are 33 measurements available. Table 3 shows the search space covered by the proposed algorithm and the number of C_{ks} found and confirmed up to $k = 10$.

As can be seen in Figs. 3 and 4, for both steps, better performance has been achieved with the GPU implementation, reaching in Step 2 an average speed-up of 13x when compared to the original serial implementation, as it can be seen in Table 4. Besides, as it is shown in Table 5, in Step 3 we achieved speed-ups of 25x.

5.2 IEEE 30-Bus System

Tests were also performed with the IEEE 30-bus system with a low redundant measurement set, adapted from [2]. Table 6 confirms that the lack of redundant measurements results in more C_{ks} of lower cardinalities. Therefore, Step 3 becomes more expensive, as those C_{ks} are added early to the solution set and are always compared to every possible C_k found in Step 2.

Table 3. IEEE 14-bus system visited combinations and C_{ks} confirmed

Cardinality k	No. of visited combinations of measures	No. of C_{ks} identified
1	33	0
2	528	13
3	5456	0
4	40920	1
5	237336	1
6	1107568	1
7	4272048	13
8	13884156	9
9	13884156	13
10	38567100	11

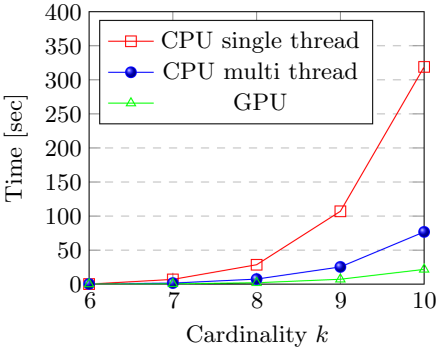

Fig. 3. Performance for Step 2, 14-bus case

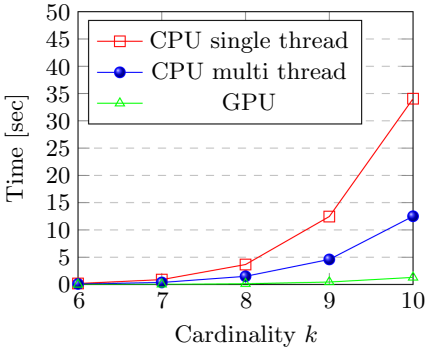

Fig. 4. Performance for Step 3, 14-bus case

Table 4. Speed-ups achieved for Step 2 in the IEEE 14-bus system

Cardinality k	Speed-up					Average
	6	7	8	9	10	
CPU Single-Thread × CPU Multi-Thread	4.31	4.32	3.83	4.23	4.16	4.17
CPU Single-Thread × GPU	10.39	13.33	13.69	14.81	14.70	13.37
Multi-Thread × GPU	2.41	3.09	3.56	3.50	3.54	3.22

Figures 5 and 6 show that GPU implementation again presented the best performance. As it is shown in Table 7 and 8, the speedups observed in Steps 2 and 3 were 13x and 18x, respectively. The decrease of the speedup in Step 3,

Table 5. Speed-ups achieved for Step 3 in the IEEE 14-bus system

Cardinality k	Speed-up					Average
	6	7	8	9	10	
CPU Single-Thread × CPU Multi-Thread	2.01	2.40	2.44	2.71	2.72	2.46
CPU Single-Thread × GPU	20.33	25.48	26.50	27.61	26.21	25.23
Multi-Thread × GPU	10.11	10.6	10.83	10.17	9.625	10.27

Table 6. IEEE 30-bus visited combinations and C_{ks} confirmed.

Cardinality k	No. of visited combinations of measurements	No. of C_{ks} identified
1	42	4
2	861	11
3	11480	19
4	111930	96
5	850668	345
6	5245786	1297
7	26978328	2911
8	118030185	8335
9	445891810	39336

concerning the one obtained with the IEEE 14-bus system, occurs because of a larger *solSet* is transferred to GPU global memory in each loop.

The analysis for k = 9 became too expensive for the single thread CPU implementation. In GPU it took around 23 min to perform Step 3.

As mentioned in the introduction section, the specialized literature is sparse in studies related to the problem of searching for critical elements is state

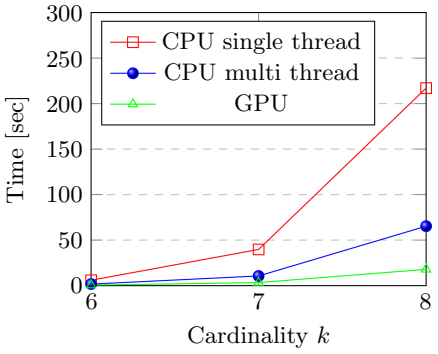

Fig. 5. Performance for Step 2, 30-bus case.

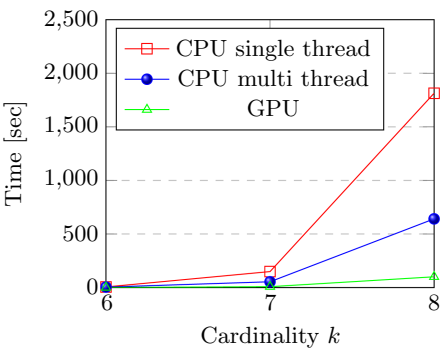

Fig. 6. Performance for Step 3, 30-bus case.

Table 7. Speed-ups achieved for Step 2 in IEEE 30-bus system

Cardinality k	Speed-up			Average
	6	7	8	
CPU Single-Thread × CPU Multi-Thread	3.41	3.75	3.31	3.51
CPU Single-Thread × GPU	11.60	12.92	13.11	12.55
Multi-Thread × GPU	3.05	3.18	3.64	3.29

Table 8. Speed-ups achieved for Step 3 in IEEE 30-bus system

Cardinality k	Speed-up			Average
	6	7	8	
CPU Single-Thread × CPU Multi-Thread	1.32	2.75	2.83	2.30
CPU Single-Thread × GPU	8.12	18.22	18.07	18.03
Multi-Thread × GPU	6.15	5.53	6.38	6.35

estimation. In [2], one can find results of a B&B algorithm implemented in a CPU environment to identify C_{ks} of cardinalities up to five in the IEEE 30-bus system. However, no information on the estimated computing time to perform this task is provided, which could enable comparative studies on the subject. At this point, it is opportune to comment that the results obtained in the paper point to a successful novel application of GPUs to perform the criticality analysis task in power system state estimation studies. As such, the proposed use of GPUs can be seen as a reference implementation, to demonstrate (proof of concept) that it is a feasible option when compared with its CPU counterparts, acting as a starting point for more elaborated (optimized) implementations. Thus, the results obtained here serve as benchmarks for future correlated research works.

6 Conclusion

In this work we developed a novel parallel approach for identification of measurement criticalities. Our solution may affect the quality of power system state estimation, due to its high increase of performance. The use of GPUs proved to be useful to address this difficult combinatorial problem, allowing the determination of cardinalities that would be very costly to achieve in a single-thread CPU. Different test systems and measurement redundancies were tested and the proposed implementation lead to speedups up to 28×, showing its potential for the identification of measurement criticalities, particularly those of high cardinality, whose determination is usually more challenging.

Since managing data transfer between host and device is a bottleneck for all the solutions, in future works, we intend to include steps 1 and 4 at the GPU level. However, adapt those steps to parallel implementation is not a trivial task. In step 1, it is challenging to build the combination matrix in GPU, because it is

not straightforward to correlate each combination to each thread identification, according to [11]. In step 4, since we implemented the solution set in a stack, it is not possible to add elements concurrently to set. We believe that exists a better data structure that ensures performance improvements in this implementation. Important improvements in the parallel code are also possible to be made, such as the usage of GPU streams in order to assist large-scale combination arrays. We also believe that we can better explore parallel approaches to assist the Branch and Bound heuristic that were already implemented for criticality analysis solution.

References

1. Abur, A., Gómez Expósito, A.: Power System State Estimation: Theory and Implementation. Dekker, New York (2004)
2. Augusto, A.A., Do Coutto Filho, M.B., de Souza, J.C.S., Guimaraens, M.A.R.: Branch-and-bound guided search for critical elements in state estimation. IEEE Trans. Power Syst. **34**, 2292–2301 (2019)
3. Augusto, A.A., Guimaraens, M.A.R., Do Coutto Filho, M.B., Stacchinni de Souza, J.C.: Assessing strengths and weaknesses of measurement sets for state estimation. In: 2017 IEEE Manchester PowerTech, pp. 1–6. IEEE, Manchester (2017)
4. Boukedjar, A., Lalami, M.E., El-Baz, D.: Parallel branch and bound on a CPU-GPU system. In: 2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 392–398. IEEE, Munich (2012)
5. Castillo, E., Conejo, A.J., Pruneda, R.E., Solares, C., Menendez, J.M.: $m - k$ Robust observability in state estimation. IEEE Trans. Power Syst. **23**, 296–305 (2008)
6. Clements, K.A., Davis, P.W.: Multiple bad data detectability and identifiability: a geometric approach. IEEE Trans. Power Deliv. **1**, 355–360 (1986)
7. Crainic, E.D., Horisberger, H.P., Do, X.D., Mukhedkar, D.: Power network observability: the assessment of the measurement system strength. IEEE Trans. Power Syst. **5**, 1267–1285 (1990)
8. Ilya, P., Pavel, C., Olga, M., Andrey, P.: The usage of parallel calculations in state estimation algorithms. In: 2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 1–5. IEEE, Phuket (2017)
9. Karimipour, H., Dinavahi, V.: Accelerated parallel WLS state estimation for large-scale power systems on GPU. In: 2013 North American Power Symposium (NAPS), pp. 1–6. IEEE, Manhattan (2013)
10. Karimipour, H., Dinavahi, V.: On false data injection attack against dynamic state estimation on smart power grids. In: 2017 IEEE International Conference on Smart Energy Grid Engineering (SEGE), pp. 388–393. IEEE, Oshawa (2017)
11. Luong, T.V., Melab, N., Talbi, E.-G.: Large neighborhood local search optimization on graphics processing units. In: 2010 IEEE International Symposium on Parallel a& Distributed Processing. Workshops and PhD Forum (IPDPSW), pp. 1–8. IEEE, Atlanta (2010)
12. Monticelli, A.: Power System State Estimation: A Generalized Approach. Kluwer, Norwell (1999)
13. Ruskey, F., Williams, A.: The coolest way to generate combinations. Discrete Math. **309**, 5305–5320 (2009)

14. Sou, K.C., Sandberg, H., Johansson, K.H.: Computing critical k-tuples in power networks. IEEE Trans. Power Syst. **27**, 1511–1520 (2012)
15. Xie, L., Mo, Y., Sinopoli, B.: False data injection attacks in electricity markets. In: 2010 First IEEE International Conference on Smart Grid Communications, pp. 226–231. IEEE, Gaithersburg (2010)