Extended Kalman Filter-Based Parallel Dynamic State Estimation

Hadis Karimipour, *Student Member, IEEE,* and Venkata Dinavahi, *Senior Member, IEEE*

***Abstract*—There is a growing need for accurate and efficient real-time state estimation with increasing complexity, interconnection, and insertion of new devices in power systems. In this paper, a massively parallel dynamic state estimator is developed on a graphic processing unit (GPU), which is especially designed for processing large data sets. Within the massively parallel framework, a lateral two-level dynamic state estimator is proposed based on the extended Kalman filter method, utilizing both supervisory control and data acquisition, and phasor measurement unit (PMU) measurements. The measurements at the buses without PMU installations are predicted using previous data. The results of the GPU-based dynamic state estimator are compared with a multithread CPU-based code. Moreover, the effects of direct and iterative linear solvers on the state estimation algorithm are investigated. The simulation results show a total speed-up of up to 15 times for a 4992-bus system.**

***Index Terms*—Compute unified device architecture (CUDA), data parallelism, dynamic state estimation (DSE), extended Kalman filter (EKF), graphic processing units (GPUs), large-scale systems, massive-thread, multithread, OpenMP, parallel programming, phasor measurement units (PMUs).**

I. Introduction

THE EVOLUTION of power systems toward the new smart grid era is bringing unprecedented operational challenges toward online monitoring of networks. Traditional dynamic state estimation (DSE) is not scalable enough to process the large amount of data generated over the grid, and is prone to computational bottlenecks. Existing DSE paradigms mainly focus on complexity reduction using partial measurements, hierarchical, and decoupled methods [1]–[4] which compromise the accuracy for speed, but are not fast enough to predict the real-time behavior of the system. Other methods that focus on estimation accuracy by increasing either modeling or algorithmic complexity (see [5]–[8]), are computationally onerous limiting their practical applicability to small scale systems. Indeed, new approaches for DSE which are both fast and accurate are required. Recent research in power system state estimation within the smart grid context is focused on the following subjects.

1) Distributed state estimation by domain decomposition to scale down the problem [9]–[11].
2) Combined parameter and DSE on practical data [12].
3) Vulnerability analysis of power grids as a cardinality minimization problem [13].
4) Secure state estimation by identification and protection of critical measurements [14], [15].

While the above methods address important issues, they overlook the computational efficiency aspect and speed of the state estimation process which are vital for real-time monitoring and control of the grid. In addition, the proposed approaches are mainly tested on CPU-based hardware for small system sizes which behave somewhat differently from large-scale power systems in terms of computational complexity. In contrast, this paper presents a massively parallel dynamic state estimator (MPDSE) utilizing extended Kalman filter (EKF) on the graphic processing unit (GPU). GPUs have the following advantages over CPU clusters.

1) *Parallelism:* Parallelization using GPU is fine grained parallelization which is a lot different from coarse grained parallelization on CPU. In contrast to the CPU with a limited number of arithmetic cores, the GPU is composed of hundreds of cores known as stream processors (SPs) that can simultaneously handle thousands of threads [16], [17].
2) *Extensibility:* Unlike CPU with limited achievable speed-up, the maximum achievable speed-up by massive parallelism in GPU is proportional to the number of cores [18].
3) *Cost:* A GPU with hundreds of core is a lot cheaper than a system with hundred CPU cores. Basically, the GPU has enormous cost advantage, GFlops per dollar, in comparison with CPUs [19].

The popularity of the GPUs in the field of high-performance computing is due to their ability to provide computational power for massively parallel problems at a reduced cost [20]. The novelty of this paper includes the data collation method which is used to prepare measurement set, and the massive-thread parallel implementation of the DSE on GPU which to the best of our knowledge is not reported yet. In this paper, using an NVIDIA GPU, separate tasks are assigned to individual compute unified device architecture (CUDA) abstracted threads. Therefore, the computationally onerous tasks are offloaded and executed in parallel utilizing thousands of threads, accelerating the process of state estimation significantly.

Manuscript received February 25, 2014; revised July 7, 2014, September 23, 2014, and November 17, 2014; accepted December 25, 2014. Date of publication January 20, 2015; date of current version April 17, 2015. This work was supported by the Natural Science and Engineering Research Council of Canada. Paper no. TSG-00147-2014.

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2V4 Canada (e-mail: dinavahi@ece.ualberta.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSG.2014.2387169

The massively parallel processing capability of the GPU has already been exploited for power system applications, such as power flow analysis, transient stability simulation, and electromagnetic transient simulation [20]–[25].

The implementation of synchronized phasor measurement unit (PMU) technology has enabled system operators to monitor the real-time network conditions [26]. PMUs which provide measurements by sampling instantaneous waveforms can deliver up to 50/60 measurements per second. For large-scale networks, installing enough PMUs for full network observation may be expensive and impractical. A more realistic and feasible deployment of the PMU for DSE is to use both conventional supervisory control and data acquisition (SCADA) and synchronized measurements together. Most of the proposed approaches [27]–[29] neglected the fact that PMU measurements are a lot faster than SCADA measurements; however, [30]–[32] interpolated states at the PMU unobservable buses considering that there are more measurements available in the buses with PMU installation. To take advantage of the PMUs faster refresh rate a lateral data collation method is investigated in this paper for the MPDSE. In the first level, we extrapolate the SCADA measurements employing the exponential moving average method. Under contingency scenarios, in case of significant difference in PMU measurements, a correction term is added to the results of prediction. Once the measurements at all buses are available, they are transformed to SCADA format before being fed to the state estimator in the second level.

The organization of this paper is as follows. The data collation and the MPDSE algorithm are presented in Section II. Section III describes the GPU and the CPU programming paradigms for DSE. Section IV presents the experimental results and analysis, followed by the conclusion in Section V.

II. Lateral Massively Parallel DSE

Using the present and previous states of the network, DSE predicts the state vector one step ahead of the time. In this section, a lateral massively parallel state estimation algorithm is formulated. The two stages of the MPDSE include the data collation and the state estimation process itself which are done in parallel, i.e., when the state is being estimated at the present time, data is being collected for the next time step, simultaneously. The data collation procedure prepares a uniform set of measurements for EKF-based state estimator.

A. Data Collation

The main goal in this section is to provide a coherent set of measurements for the MPDSE algorithm. Since it is not possible to make the whole system observable using PMUs due to the high cost of this technology, the proposed data collation method extrapolates SCADA measurements to update them as fast as PMU measurements arrive. The process of data collation is done under the following assumptions.

1) The network is observable with the existing SCADA measurements, and the PMU devices are installed at the generator buses.
2) The number of PMU channels is limited to a maximum of two (one for voltage and one for current) for economical reasons.
3) Since practical state estimation update every 30–60 s, and SCADA measurements update every 2–5 s [33], [34], SCADA and PMU refresh rate are considered as every 2 and 30 s, respectively.
4) Based on the previous assumption, in between two SCADA measurements 60 PMU measurements are available. Since the changes for close measurements are very small, the buffer length of six is chosen to use the average of each six consecutive PMU measurements. So, in between two SCADA measurements there are ten PMU measurements. However, under contingency scenarios, the actual refresh rate of the PMU measurements is used to decide the condition of the system.
5) For simplicity the measurement uncertainty due to instrument transformers [35] is neglected, and PMU measurements are considered with higher accuracy than the SCADA measurements, by assigning higher weights to them in the error covariance matrix.
6) SCADA data are recorded with local time stamps, so the time skew between SCADA and PMU measurements will be negligible considering the fact that quantities provided by SCADA measurements do not change significantly in a short time interval.
7) The time step Δt is equal to 0.2 s which is the time interval between two averaged PMU measurements.
8) For simplicity it is assumed that the reference of PMU measurements which is associated with the GPS timing signal is equal to the slack bus angle.

The overall measurement set (m) is divided into two subsets: PMU measurements ($\mathrm{m^P}$) and SCADA measurements ($\mathrm{m^S}$). Since the refresh rate of $\mathrm{m^P}$ is a lot faster than $\mathrm{m^S}$, there are more measurements available for the buses with PMU installations during the same time interval. To take advantage of all available measurements, missing SCADA measurements are extrapolated employing the exponential moving average method [36]. This method assigns different weights to the previous measurements in a way that the old measurements fade exponentially and new measurements have more effect on the result of prediction as follows:

$$\tilde{\mathrm{m}}^{\mathrm{E}}_{k+1} = \mu \mathrm{m}^{\mathrm{S,E}}_{k} + (1-\mu)\tilde{\mathrm{m}}^{\mathrm{E}}_{k}, \quad 0 < \mu < 1 \tag{1}$$

where μ is the scalar smoothing constant, which is chosen to be 0.7 in this paper. $\mathrm{m}^{\mathrm{S,E}}_{\mathrm{k}}$ and $\tilde{\mathrm{m}}^{\mathrm{E}}_{k}$ are previous measurements (including actual and extrapolated SCADA measurements) and extrapolated measurements, respectively. The indices $\mathrm{k} = \mathrm{t}$ and $\mathrm{k}+1 = \mathrm{t}+\Delta\mathrm{t}$ are used for present time and one step in the future, respectively.

To extrapolate a SCADA measurement, the last ten available measurements which contain both measured SCADA and extrapolated SCADA are used. The algorithm, extrapolates all measurements even those that arrive exactly at the time the new set of SCADA measurements arrive. It should be noted that in every ten extrapolated SCADA measurements, two of them will be available by actual measurement. Whenever new SCADA is available, extrapolated measurement is replaced by

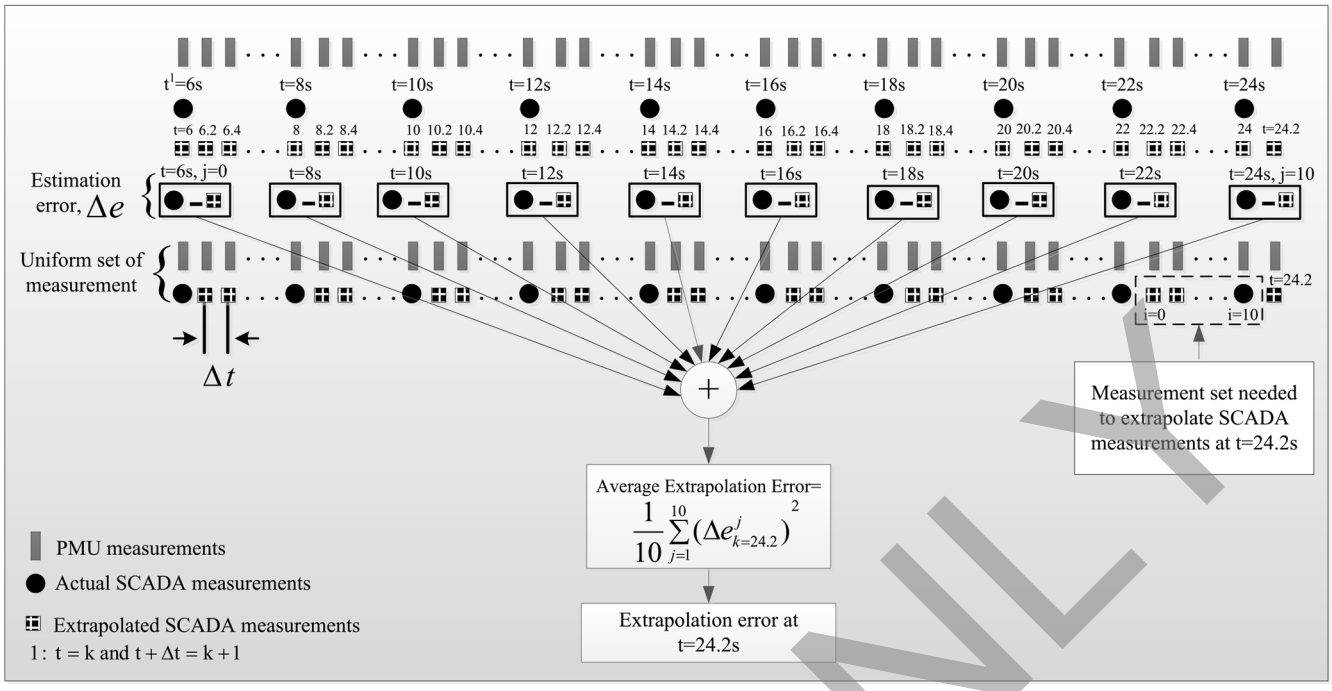

Fig. 1. Example of data collation for MPDSE.

the actual measurement. To increase the accuracy, the average of last ten estimation errors (difference between actual SCADA measurements and the extrapolated measurements) are added to the predicted value. Expanding $\tilde{m}^{E}_{k}$ with its components results in the following:

$$\tilde{m}^{E}_{k+1} = \mu \sum_{i=0}^{10} (1-\mu)^{i} m^{S,E}_{k-i} + (1-\mu)^{11} \tilde{m}^{E}_{k-11} + \frac{1}{10} \sum_{j=1}^{10} \left(\Delta e^{j}_{k}\right)^2 \quad (2)$$

where Δe^{j}_{k} represents the estimation error at the time instant k. Indices i and j refer to the last ten available measurements and estimation errors, respectively. An example of data collation for t = 24.2 s is shown in Fig. 1.

In order to have a uniform set of measurements, the PMU measurements which are in polar format are transformed into cartesian format (m^{P}_{T} in Fig. 2). For a given function f(r, φ) in polar coordinates the relationship between derivatives in cartesian ($z = r\cos(\varphi), y = r\sin(\varphi)$) and polar ($r = \sqrt{z^2 + y^2}, \varphi = \arctan(y/z)$) coordinates is as follows:

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial r}\frac{\partial r}{\partial z} + \frac{\partial f}{\partial \varphi}\frac{\partial \varphi}{\partial z} = \cos(\varphi)\frac{\partial f}{\partial r} - \frac{\sin(\varphi)}{r}\frac{\partial f}{\partial \varphi}$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial f}{\partial \varphi}\frac{\partial \varphi}{\partial y} = \sin(\varphi)\frac{\partial f}{\partial r} + \frac{\cos(\varphi)}{r}\frac{\partial f}{\partial \varphi}. \quad (3)$$

The error covariance matrix R corresponding to PMU measurement errors in polar coordinates must also be transformed to cartesian coordinates. By definition of differentiability the incremental change Δz is given as

$$\Delta z = z(r + \Delta r, \varphi + \Delta\varphi) - z(r, \varphi) \simeq \frac{\partial z}{\partial r}\Delta r + \frac{\partial z}{\partial \varphi}\Delta\varphi. \quad (4)$$

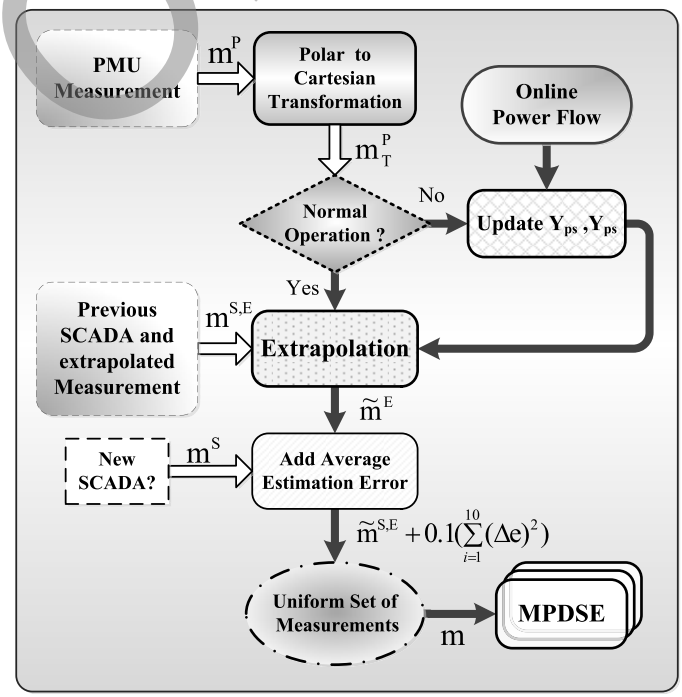

Fig. 2. Data collation process flowchart.

Similarly, the incremental change Δy can also be defined resulting in the following general transformation:

$$\begin{bmatrix}\Delta z \\ \Delta y\end{bmatrix} = \begin{bmatrix}\cos(\varphi) & -r\sin(\varphi) \\ \sin(\varphi) & r\cos(\varphi)\end{bmatrix}\begin{bmatrix}\Delta r \\ \Delta\varphi\end{bmatrix} = [T_r]\begin{bmatrix}\Delta r \\ \Delta\varphi\end{bmatrix}. \quad (5)$$

Based on definition of covariance matrix [37]

$$R(z, y) = Ex\left[(z - \bar{z})\,(y - \bar{y})^{T}\right] \quad (6)$$

where Ex indicate the expected value. $\bar{z}$ and $\bar{y}$, represent the mean value of z and y, respectively. Substituting (5) in (5) using the transformation matrix (T_r), the error covariance sub-matrix of PMU measurements in cartesian format can be driven as follows:

$$\begin{aligned} R^{C}_{PMU} &= R(\Delta z, \Delta y) = Ex\left[(T_r \Delta r)(T_r \Delta \varphi)^T\right] \\ &= Ex\left[T_r \Delta r \Delta \varphi^T T_r^T\right] = T_r Ex\left[\Delta r \Delta \varphi^T\right] T_r^T \\ &= T_r R(\Delta r \Delta \varphi) T_r^T = T_r R^{P}_{PMU} T_r^T. \end{aligned} \tag{7}$$

$\bar{\Delta} z$ and $\bar{\Delta} y$ assumed to be zero.

Superscripts C and P refers to cartesian and polar format, respectively. Since the PMU measurements are assumed to have higher accuracy than SCADA measurements, more weight is assigned to them in measurement error covariance matrix

$$R = \begin{bmatrix} R^{C}_{SCADA} & 0 \\ 0 & R^{C}_{PMU} \end{bmatrix}. \tag{8}$$

In case of sudden changes in the PMU measurements the accuracy of the estimation may degrade. By online tracking (as fast as measurement update rate) of the difference between two consecutive PMU measurements at the buses with PMU installations, the algorithm decides whether the network was in normal condition or not. The threshold for detecting contingency is set to 50% of previous measurement; it is assumed that if the next PMU measurement changes more than 50% of the previous value there is a contingency. The proposed method also can be applied for smaller thresholds. To handle the contingency effect on the state estimation process, a correction step is added to the algorithm. Nodal equations in a power network can be written as

$$\begin{bmatrix} I_P \\ I_S \end{bmatrix} = \begin{bmatrix} Y_{PP} & Y_{PS} \\ Y_{SP} & Y_{SS} \end{bmatrix} \begin{bmatrix} V_P \\ V_S \end{bmatrix} \tag{9}$$

where the subscripts P and S refer to buses with PMU measurement and SCADA measurement subsets, respectively. In general

$$\Delta V_P = (\Delta Y_{PP})^{-1} (\Delta I_P - \Delta Y_{PS} \Delta V_S). \tag{10}$$

From (10), it can be concluded that changes in ΔV_S is proportional to changes in ΔV_P. Therefore, in the case of sharp or sudden changes in the PMU measurements, only Y_{PS} and Y_{PP} needed to be updated using online power flow. These matrices are too small compare to Y_{SS}. Fig. 2 shows the block diagram of the entire data collation process.

B. Massively Parallel DSE

The generic power system for DSE can be described by

$$x_{k+1} = f(x_k) + w_k \tag{11}$$

$$m_{k+1} = h(x_{k+1}) + \varepsilon_{k+1}, \quad \varepsilon_k \sim N(0, R_k) \tag{12}$$

where x is a vector of system states comprising of voltage magnitudes and phase angles at all buses except the slack bus where $V_1 = 1\angle 0°$ p.u. Since the phase angle in slack bus is considered 0, there are $2n-1$ states to be estimated. f(x), m, and h(x), are vectors of nonlinear system transition function, unified measurements, and nonlinear measurement functions, respectively. For a system with n buses and m lines, there are $2m + 2n + 1$ elements in each measurement vector: $2m$ power flows, $2n$ power injections, and slack bus measurements. ε and w are measurements and system noises assuming normal distribution with zero mean, and R is the $(2m + 2n + 1) \times (2m + 2n + 1)$ measurement error covariance matrix. Equation (11) can be linearized as follows if the time frame is small enough:

$$x_{k+1} = F_k x_k + a_k + \omega_k, \quad \omega_k \sim N(0, Q_k) \tag{13}$$

where F_k represents the $(2n-1) \times (2n-1)$ state transition matrix between two time frames, a_k is the vector of associated behavior of the state trajectory, and ω_k is the Gaussian noise vector with zero mean and covariance matrix Q_k. The MPDSE utilizing EKF is composed of three major steps: 1) identification; 2) prediction; and 3) filtering.

1) Parameter Identification: To evaluate the dynamic model, unknown parameters need to be calculated online. Holt's exponential smoothing technique [38] was used for identification of F_k and a_k. Based on this method, F_k and a_k can be described as follows: if $\tilde{x}$ and $\hat{x}$ represent the predicted and estimated value of the states, respectively:

$$\begin{aligned} F_k &= \alpha(1+\beta) I_{idn}, \quad 0 < (\alpha, \beta) < 1 \\ a_k &= (1+\beta)(1-\alpha)\tilde{x}_k - \beta \gamma_{k-1} + (1+\beta)\xi_{k-1} \\ \gamma_k &= \alpha \hat{x}_k + (1-\alpha)\tilde{x}_k \\ \xi_k &= \beta(\gamma_k - \gamma_{k-1}) + (1-\beta)\xi_{k-1} \end{aligned} \tag{14}$$

where α and β are smoothing parameters. Under normal operation conditions it is possible to adjust F_k and a_k such that Q_k remains constant. I_{idn} is the $(2n-1) \times (2n-1)$ identity matrix. However, considering the dynamic behavior of the network neglecting the changes in Q_k may result in inaccurate prediction. Online estimation of Q_k can be formulated as [39]

$$\hat{Q}_{k+1} = Q_k \sqrt{\frac{\text{trace}\left\{H_{k+1}\left(F_k \rho_k F_k{}^T + \hat{Q}_k\right) H^T{}_{k+1}\right\}}{\text{trace}\left\{H_{k+1}\left(F_k \rho_k F_k{}^T + Q_k\right) H^T{}_{k+1}\right\}}}. \tag{15}$$

2) State Prediction: Using the measurement and estimated states at the time instant k, the predicted value $\tilde{x}_{k+1}$ can be formulated as

$$\begin{aligned} \tilde{x}_{k+1} &= F_k \hat{x}_k + a_k, \quad \left(x_k - \hat{x}_k\right) \sim N(0, \rho_k) \\ \tilde{\rho}_{k+1} &= F_k \rho_k F_k^T + \hat{Q}_k, \quad (x_k - \tilde{x}_k) \sim N(0, \tilde{\rho}_k) \end{aligned} \tag{16}$$

where ρ and $\tilde{\rho}$ are $(2n-1) \times (2n-1)$ error covariance matrices for estimated and predicted values, respectively. The objective function J(x) was chosen to minimize both estimation and prediction errors

$$\begin{aligned} J(x) = \arg\min_x\ & [m - h(x)]^T R^{-1} [m - h(x)] \\ & + \left[x - \tilde{x}\right]^T \tilde{\rho}^{-1} \left[x - \tilde{x}\right]. \end{aligned} \tag{17}$$

The following equation satisfies the first-order optimality condition at the minimum of J(x):

$$g(x) = H^T(x) R^{-1} [m - h(x)] - \tilde{\rho}^{-1} \left[x - \tilde{x}\right] = 0 \tag{18}$$

where g(x) is the $(2n-1)\times 1$ vector of gradient of the objective function, and $\mathrm{H}=\partial \mathrm{h}/\partial \mathrm{x}$ is the $(2m+2n+1)\times(2n-1)$ Jacobian matrix. Using Taylor's expansion of h(x) around $\tilde{\mathrm{x}}_0$ (18) can be expressed as follows:

$$\begin{aligned}\mathrm{G(x)}\Delta\mathrm{(x)} &= \mathrm{H}^{\mathrm{T}}(\tilde{\mathrm{x}})\,\mathrm{R}^{-1}\left[\mathrm{m}-\mathrm{h}(\tilde{\mathrm{x}})\right]\\ \mathrm{G(x)} &= \mathrm{H}^{\mathrm{T}}(\tilde{\mathrm{x}})\,\mathrm{R}^{-1}\mathrm{H}(\tilde{\mathrm{x}})+\tilde{\rho}^{-1}\end{aligned} \tag{19}$$

where $\Delta(\mathrm{x})=\hat{\mathrm{x}}-\tilde{\mathrm{x}}_0$ is the $(2n-1)\times 1$ state mismatch vector and $\mathrm{G(x)}=\partial \mathrm{g}/\partial \mathrm{x}$ is $(2n-1)\times(2n-1)$ is the gain matrix. The state estimation algorithm given by (17)–(19) can be solved iteratively until convergence of Δ(x) to a specified threshold.

3) State Filtering: This step updates the predicted values using the next set of measurements at the time instant *k+1*. The updated state through EKF can be written as

$$\begin{aligned}\hat{\mathrm{x}}_{\mathrm{k+1}} &= \tilde{\mathrm{x}}_{\mathrm{k+1}}+k_{\mathrm{k+1}}\left(\mathrm{m}_{\mathrm{k+1}}-\mathrm{h}\left(\tilde{\mathrm{x}}_{\mathrm{k+1}}\right)\right)\\ k_{\mathrm{k+1}} &= \tilde{\rho}_{\mathrm{k+1}}\mathrm{H}^{\mathrm{T}}{}_{\mathrm{k+1}}\left[\mathrm{H}_{\mathrm{k+1}}\tilde{\rho}_{\mathrm{k+1}}\mathrm{H}^{\mathrm{T}}{}_{\mathrm{k+1}}+\mathrm{R}\right]^{-1}\\ \rho_{\mathrm{k+1}} &= \tilde{\rho}_{\mathrm{k+1}}-k_{\mathrm{k+1}}\mathrm{H}_{\mathrm{k+1}}\tilde{\rho}_{\mathrm{k+1}}\end{aligned} \tag{20}$$

where K is the $(2n-1)\times(2n-1)$ Kalman gain matrix. For the same reasons mentioned earlier, this step is also a good candidate for parallelization.

C. Extraction of Parallelism

In the proposed MPDSE method several aspects of parallelism are combined to utilize the full capability of GPUs as efficiently as possible. First, all initialization and data collation are done on the CPU. After that all of the data are transferred to the GPU for executing the MPDSE algorithm. The following types of parallelism are used in this paper.

1) *Task Parallelism:* In this level, the traditional serial algorithm is converted into various smaller and independent tasks which can be solved in parallel. All of the independent tasks in the three main steps of EKF are calculated in parallel to accelerate the algorithm. In the parameter identification, $\mathrm{a_k}$, λ_k, and $\mathrm{Q_k}$ do not rely on each other's result, so they are calculated in parallel to accelerate the algorithm. In the state prediction stage, $\tilde{\mathrm{x}}_{\mathrm{k+1}}$, $\tilde{\rho}_{k+1}$, G(x), and g(x) are parallelizable. Finally in the state filtering step, $\hat{\mathrm{x}}_{\mathrm{k+1}}$ and ρ_{k+1} can be calculated simultaneously.
2) *Data Parallelism:* This level employs the fine-grained type of parallelism that can be used on the single instruction multiple data (SIMD)-based architectures such as GPUs for the basic computations in the algorithm. Generally, matrix-vector and matrix-matrix products are time consuming for large data-sets. There are several independent for loops in the implementation of each matrix-matrix and matrix-vector products which make them the best candidates for parallelization utilizing GPU threads. Assigning each iteration in a loop to individual threads, the task can be executed in parallel by converting into a kernel. In the MPDSE algorithm, the computation of $\mathrm{a_k}$ for state prediction and G(x), K, λ, ρ and $\mathrm{H^T(x)R^{-1}H}+\tilde{\rho}^{-1}$ for state filtering can take a long time to complete even on CPU clusters. These separate tasks are composed of matrix-matrix and matrix-vector product or summations which can be assigned to an individual kernel to run in parallel. Each kernel is responsible for the calculation of that specific task. As the number of independent threads is a lot more than the CPU cores, this type of parallelization is not possible on the CPU.
3) *Parallelism in Linear Solver:* Solution of Δ(x) by inversion of G(x) is considerably expensive due to the sheer size of the inverted matrix. Two alternatives, LU decomposition as a direct method and preconditioned conjugate gradient (PCG) [40] as an iterative method were used in this paper. For iterative solvers, the preconditioner is the most challenging part to parallelize. Coarse grained or task parallelism is difficult to achieve on LU and Cholesky factorization due to inherent sequentiality. However, underlying implementation of these algorithms (vector updates, inner products, and matrix vector products) takes advantage of the data parallelism. So, these tasks are done as a combination of sequential and parallel computations.

Sparse matrix-vector multiplication and sparse triangular solve is used for GPU implementation using cuSPARSE library [41]. Fig. 3 shows the overall MPDSE flowchart, and Table I summarizes the sequential and parallel variables.

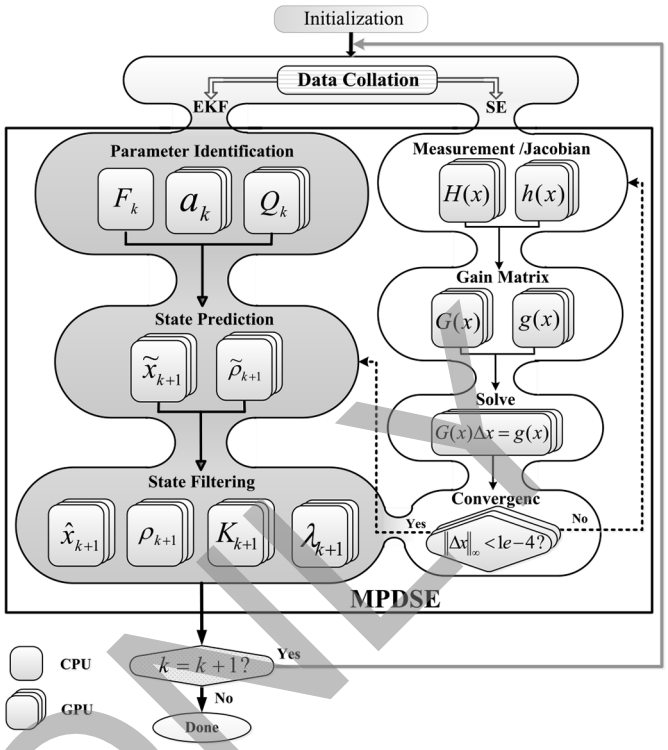

Fig. 3. MPDSE operation flowchart.

TABLE I
SUMMARY OF SEQUENTIAL AND PARALLEL VARIABLES IN MPDSE

Task	Sequential	Task Parallel	Data Parallel
Initialization	Y, R,...	–	–
Data Collation	$\tilde{\mathrm{m}}^{\mathrm{E}}$	–	–
Parameter Identification	F	a, λ, Q	a, λ, Q, ξ
State Prediction	H, h	$\tilde{\mathrm{x}}$, $\tilde{\rho}$	H, h,$\tilde{\mathrm{x}}$, $\tilde{\rho}$
State Filtering	–	$\hat{\mathrm{x}}$, ρ	$\hat{\mathrm{x}}$, ρ, λ K
Linear solver	LU, PCG	–	G, g, Δx, LU, PCG

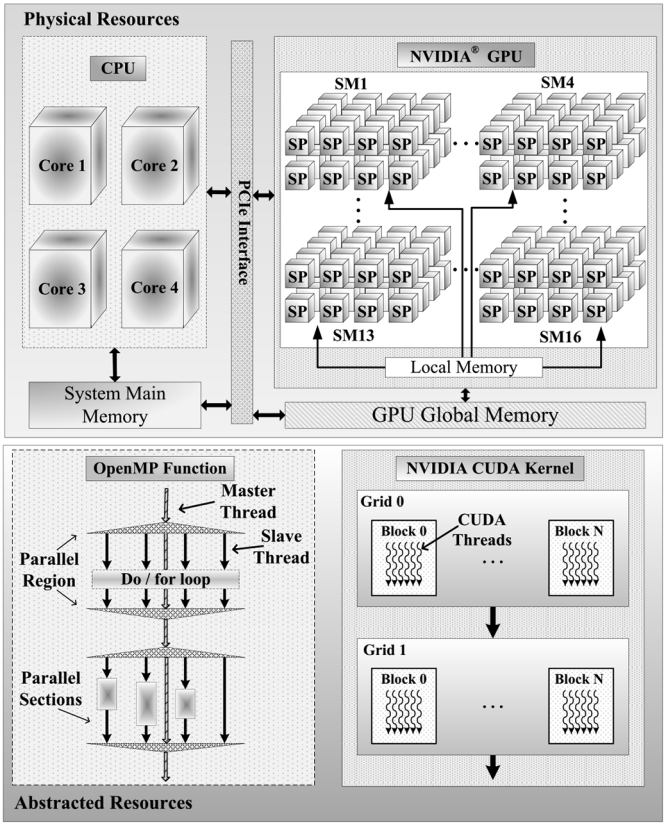

Fig. 4. CUDA and OpenMP physical and abstracted resources.

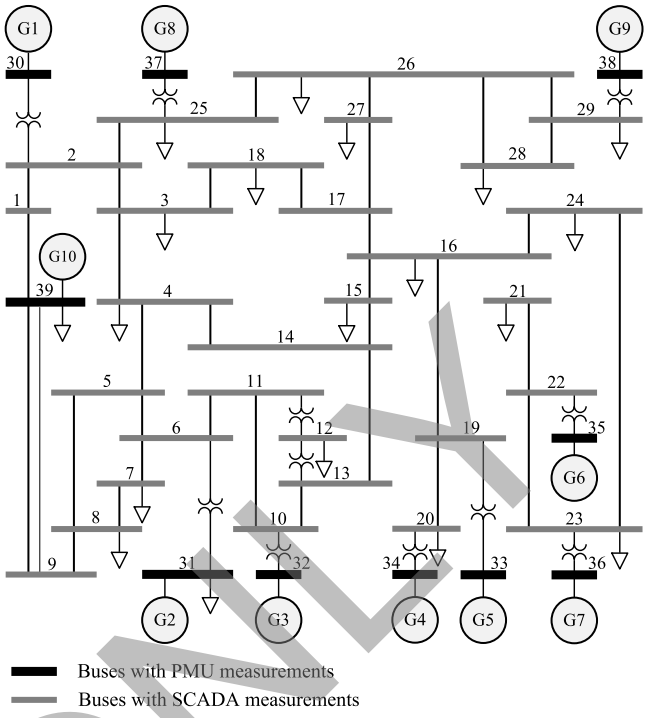

Fig. 5. IEEE 39-bus power system used to build the large test cases.

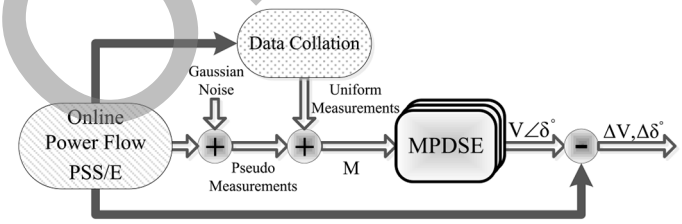

Fig. 6. MPDSE test procedure.

III. GPU AND CPU SYSTEM ARCHITECTURE

In contrast to the CPU with a limited number of arithmetic cores, capable of processing a few heavy-weight threads in parallel, the GPU is composed of hundreds of cores known as SPs that can simultaneously execute thousands of light-weight threads using the SIMD paradigm.

A. Hardware Setup

The hardware used in this paper is one unit of Tesla S2050 GPU from NVIDIA with 148 GB/s memory bandwidth. It has 448 cores which deliver up to 515 Gigaflops of double-precision peak performance. This device contains 14 streaming multiprocessors (SMs), each with 32 SPs, an instruction unit, and on-chip memory [42]. CUDA version 5.0 with compute capability 2.0 is used for programming. The CPU is the quad-core Intel Xeon E5-2620 with 2.0 GHz core clock and 32 GB memory with 42.6 GB/s memory bandwidth, running 64-bit Windows 7 operating system.

B. CUDA and OpenMP Abstractions

Fig. 4 shows the physical and abstracted resources in a CPU and a GPU. CUDA is the general-purpose programming model for the NVIDIA GPU hardware [43]. The GPU runs its own functions called kernels independently under the CPUs control. Typically, when a kernel is called, a fixed number of threads execute the same kernel in parallel. There is a two-level hierarchy in each thread namely, blockId and threadId which distinguish the specific portion of the data to process. The top level is a 2-D array of blocks which are organized as a grid. All blocks in a grid have the same dimensions and share the same blockId. While a generic programming language such as OpenCL [44] could have been used to program GPU, CUDAs advantages in providing advanced debugging and profiling tools, better performance, and higher level of abstraction are the main reason for its adoption in this paper.

OpenMP is a standard application programming interface (API) for multicore CPUs, which does not require major code reformation for parallelization. It also includes environment variables to facilitate scheduling and parallelism at runtime [45]. The program begins as a single process called a master thread which executes sequentially. The master thread creates a group of slave threads within the parallel construct. At the end of the construct only the master thread remains while the rest of the slave threads synchronize and terminate.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

To evaluate accuracy and efficiency of the parallel DSE algorithms, experiments were conducted based on two separate simulation codes: multithread CPU-based code in C++, and a massive-thread GPU-based code written in C++ and CUDA. The results of state estimation were compared utilizing

TABLE II
SUMMARY OF OVERALL ESTIMATION TIME FOR MULTITHREAD AND MASSIVE THREAD DSE UNDER CONTINGENCY CONDITION

Case	No. of buses	No. of meas.	Jacobian H(x)	Gain G(x)	Cond. no. CG	Cond. no. PCG	T_{Ex}^{CPU} LU	T_{Ex}^{GPU} LU	T_{Ex}^{CPU} PCG	T_{Ex}^{GPU} PCG	S_p LU	S_p PCG
1	39	171	171×77	77×77	2.4E+02	1.6E+01	0.49s	0.29s	0.38s	0.19s	1.69	2
2	78	347	347×155	155×155	9.7E+03	6.3E+02	1.16s	0.59s	0.83s	0.39s	1.96	2.12
3	156	699	699×311	311×311	3.9E+04	1.1E+03	4.5s	2.04s	3.1s	1.1s	2.2	2.81
4	312	1421	1421×623	623×623	8.9E+04	1.8E+03	19.2s	6.8s	13.4s	4.3s	2.8	3.02
5	624	2865	2865×1247	1247×1247	2.8E+05	4.9E+04	45.3s	12.5s	38.8s	9.8s	3.6	3.9
6	1248	5825	5825×2495	2495×2495	3.6E+06	2.6E+05	146s	26s	109.1s	20s	5.6	5.4
7	2496	11553	11553×4991	4991×4991	2.4E+07	1.3E+06	369s	40s	290.4s	33s	9.2	8.8
8	4992	23151	23151×9983	9983×9983	4.5E+08	9.4E+06	932s	59s	722.5s	48s	15.9	15.05

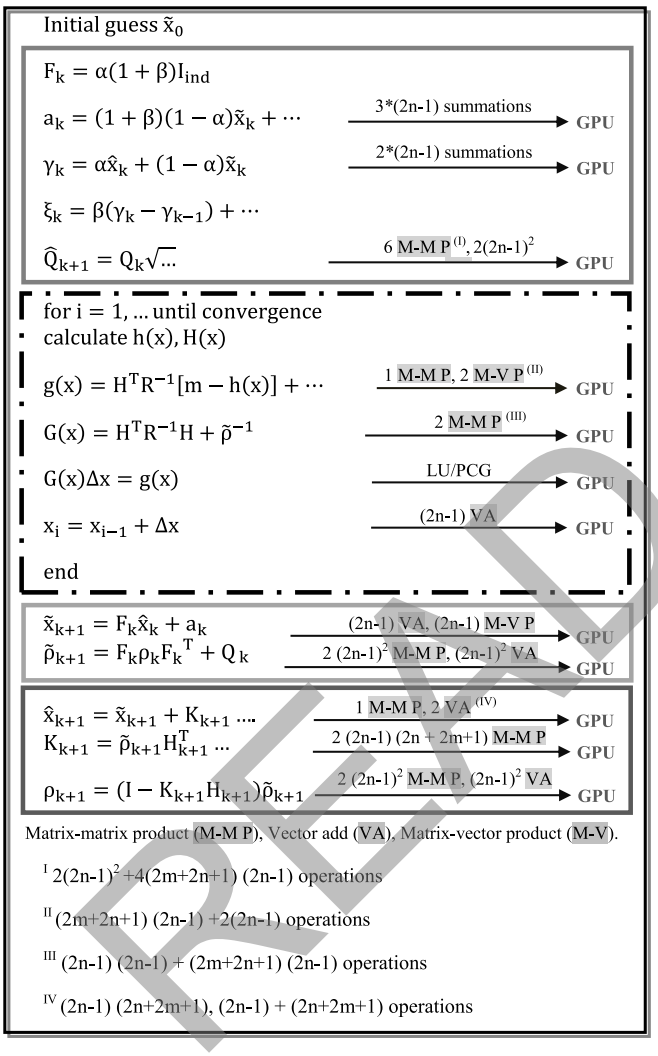

Fig. 7. Pseudo code for GPU implementation of MPDSE.

both iterative PCG and direct LU decomposition methods for the linear solver. Since the matrices are highly sparse in state estimation, all matrices and vectors are stored in compressed sparse row format to reduce the computational burden.

Large-scale test power systems were constructed to explore the efficiency of the GPU-based MPDSE program. Case 1 is the IEEE 39-bus system (Fig. 5) which was duplicated and interconnected to create large-scale systems, whose dimension n is obtained as $n = 39 * 2^{S_c}$, where $S_c = \Gamma - 1$ with Γ being the test case index. The block diagram for the MPDSE test procedure is shown in Fig. 6. Buses with PMU measurements and SCADA measurements in the IEEE 39-bus system are shown in Fig. 5.

The uniform set of measurements which are the input to the MPDSE code are obtained by corrupting online power flow results of the test power systems with Gaussian noise of zero mean and covariance R, and combining them with the output from the data collation process. Therefore, to assess the accuracy of the state estimator, the results are verified using bus voltage magnitudes and phase angles for all test cases modeled in PSS/E. While it is possible to use a partial set of measurement, all of the measurements are included to make the problem as complicated as possible for the GPU. For each time step the measurement set and all of the steps in Fig. 3 except state filtering are calculated one step ahead of the time, so the MPDSE algorithm can dynamically estimate the states of the system. The simulations were done using the test data sets listed in Table II, with a tolerance of 10^{-4} for convergence of the estimated parameters.

A. Massive-Thread GPU and Multithread CPU Implementation

The communication overhead in data transfer between the CPU (host) and the GPU (device) is an important bottleneck in massive-thread programming. In the proposed GPU implementation most of the computational steps are moved to GPU to avoid any unnecessary communication between host and device. Fig. 7 illustrates the implementation of the MPDSE on the GPU. The OpenMP standard was utilized to develop the multicore DSE code. All the for loops and parallel sections were assigned to separate threads and cores. Each thread is responsible for specific portion of the tasks to execute on that core. The entire DSE task was divided into several subsets to distribute an equal workload among the threads which equals the number of the cores. Fig. 8 shows a sample of the multithread CPU code.

B. Effect of Direct and Iterative Linear Solvers

The total execution time under contingency scenario including sequential data collation, contingency check, admittance

```
#pragma omp_set_num_threads(4)
  Initial guess x̃_0

#pragma omp parallel sections
   #pragma omp section
            F_k = α(1 + β)I_ind
   #pragma omp section
      #pragma omp parallel shared (…), private (…)
            a_k = ⋯
   #pragma omp section
            γ_k = αx̂_k + (1 − α)x̃_k
ξ_k = β(γ_k − γ_{k−1}) + ⋯
#pragma omp parallel shared
   Q̂_{k+1} = Q_k√…

while ‖Δx‖_∞ < 1e − 4
 #pragma omp parallel for
          calculate h(x), H(x)
#pragma omp parallel sections
   #pragma omp section
            g(x) = H^T R^{−1}[m − h(x)] + ⋯
   #pragma omp section
            G(x) = H^T R^{−1} H + p̃^{−1}
#pragma omp parallel for
   G(x)Δx = g(x)
#pragma omp parallel for reduction (+: …)
   x_i = x_{i−1} + Δx

#pragma omp parallel sections
   #pragma omp section
            x̃_{k+1} = F_k x̂_k + a_k
   #pragma omp section
            p̃_{k+1} = F_k ρ_k F_k^T + Q_k

#pragma omp parallel for
   x̂_{k+1} = x̃_{k+1} + K_{k+1} ....
#pragma omp parallel for
   K_{k+1} = p̃_{k+1} H^T_{k+1} ...
#pragma omp parallel for
   ρ_{k+1} = (I − K_{k+1} H_{k+1}) p̃_{k+1}
```

Fig. 8. Pseudo code for quad-core CPU implementation of DSE.

matrix update, and state estimation is reported in Table II. The results obtained using double precision (64 bit) format for both CPU ($T_{\text{Ex}}^{\text{CPU}}$) and GPU ($T_{\text{Ex}}^{\text{GPU}}$) codes as the system size increased. The execution time for GPU-based program includes both execution and communication time. An incomplete Cholesky PCG [40] iterative algorithm was used to condition the gain matrix which reduced the number of iterations in each solution. Since the gain matrix G(x) is asymmetric, it is not possible to directly use the Cholesky factorization. At first, gain matrix is divided into two matrices M and N where

$$\begin{aligned} \mathrm{G} &= \mathrm{M} - \mathrm{N} = \mathrm{G}^{\mathrm{T}} - \left(\mathrm{G}^{\mathrm{T}} - \mathrm{G}\right) \\ &= \mathrm{G}^{\mathrm{T}} - \left[\left(\mathrm{G}^{\mathrm{T}} - \mathrm{G}\right)_{\mathrm{Pos}} + \left(\mathrm{G}^{\mathrm{T}} - \mathrm{G}\right)_{\mathrm{Neg}}\right] \\ &= \left[\mathrm{G}^{\mathrm{T}} - \left(\mathrm{G}^{\mathrm{T}} - \mathrm{G}\right)_{\mathrm{Pos}}\right] - \left[\left(\mathrm{G}^{\mathrm{T}} - \mathrm{G}\right)_{\mathrm{Neg}}\right]. \end{aligned} \tag{21}$$

$\mathrm{M} = [\mathrm{G}^{\mathrm{T}} - (\mathrm{G}^{\mathrm{T}} - \mathrm{G})_{\mathrm{Pos}}] = \mathrm{L}^{\mathrm{T}}\mathrm{L}$ is a symmetric positive definite (SPD) matrix which is used as a preconditioner. $(\mathrm{G}^{\mathrm{T}} - \mathrm{G})_{\mathrm{Pos}}$ is the matrix of positive element in $(\mathrm{G}^{\mathrm{T}} - \mathrm{G})$.

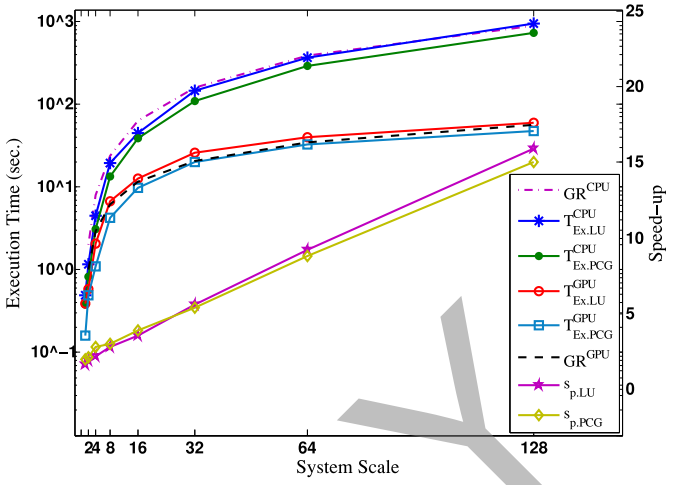

Fig. 9. Execution time (T_{Ex}) and speed-up (S_p) comparisons of multithread and massive-thread DSE along with growth rate functions.

Thus the following equation can be solved instead of (19):

$$\mathrm{L}^{-\mathrm{T}}\mathrm{G}(\mathrm{x})\check{\Delta}\mathrm{x} = \mathrm{L}^{-\mathrm{T}}\mathrm{g}(\mathrm{x}) \tag{22}$$

where $\check{\Delta}\mathrm{x} = \mathrm{L}^{-1}\Delta\mathrm{x}$. The condition numbers before and after preconditioning are shown in Table II. The results also show that the execution time reduced by approximately 20%–50% using PCG compared to the LU decomposition method. The reason is that direct methods calculate an exact solution on a single iteration, so that they deal with large size matrices which take more time to be solved; however, the iterative method starts from an initial guess and improves the solution in each iteration. Fig. 9 shows the results of comparison between the CPU and GPU simulations along with the speed-up of the parallel code for both iterative and direct solvers. As can be seen, PCG converged faster than the LU decomposition algorithm but the speed-up ($S_p = T_{\text{Ex}}^{\text{CPU}}/T_{\text{Ex}}^{\text{GPU}}$) using GPU is almost the same for both methods. The reason is that the execution times for both CPU-based PCG and GPU-based PCG experience a similar drop in each case study. Therefore, the overall speed-up is almost the same. The accuracy was mostly the same for both linear solvers, so the details are omitted from Table II.

C. Accuracy Analysis of the GPU-Based MPDSE

The performance of the proposed MPDSE method, was evaluated under both normal and contingency conditions. The estimated states in case 1 for a temporary fault at bus 10 at $t = 10$ min for a duration of 0.1 s. are shown in Fig. 10. From Fig. 11 it is clear that the proposed MPDSE can accurately track the system dynamics. The average error of 0.07 p.u. for voltage magnitudes and 2.5 rad. for phase angles during the first 5 min was due to the fact that the estimator needed to predict missing SCADA measurements at the first level (Fig. 1). Since at the beginning there was not enough history data to be used for prediction, the error of estimation was large. However, in a long term simulation neglecting the effect of incomplete historical data the average estimation error for voltage magnitudes and phase angles was 0.002 p.u. and 0.05 rad., respectively. A snapshot of the estimation error at buses

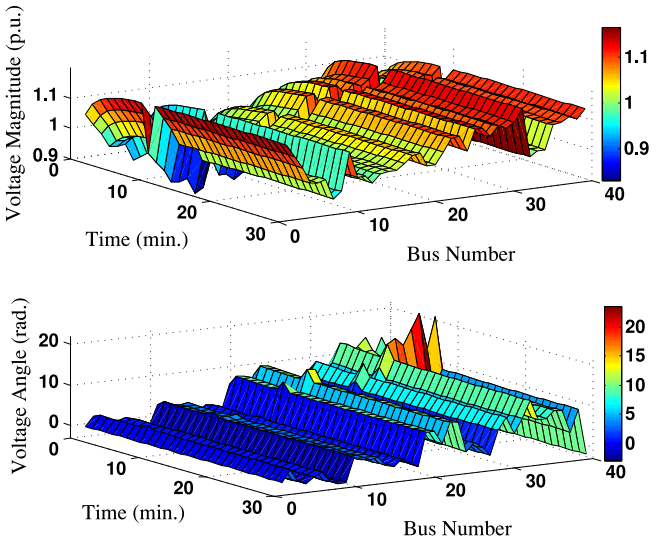

Fig. 10. Voltage magnitudes and phase angles for case 1 under faulted condition.

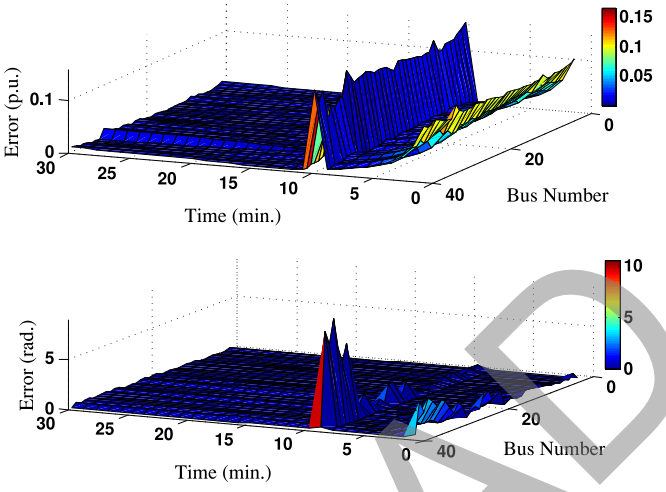

Fig. 11. Estimation errors in MPDSE for case 1 compared to PSS/E under fault conditions.

numbers 10, 11, 13, and 32 is provided in Fig. 12. The small differences compared to PSS/E results are justifiable considering the fact that the order of block execution in each GPU grid is undefined in kernel definition. Therefore, it leads to slightly different results if different CUDA blocks perform calculations on overlapping portions of data. In addition, the performance of the proposed state estimation method is tested for various number of PMU installations. The normalized Euclidian norm of the state estimation is defined as a factor to evaluate the accuracy using

$$x_{EN} = \frac{\|x - \hat{x}\|}{\sqrt{\dim(x)}} \tag{23}$$

where x_{EN} is the normalized Euclidian norm of the estimation error, and dim(x) is the dimension of vector x. x and $\hat{x}$ are vector of true states and estimated states, respectively. Table IV shows the accuracy index for both voltage magnitude and phase angle using PMU installation (N_P) at 10%, 20% and 40% of the total system buses (N_T). N_P is rounded

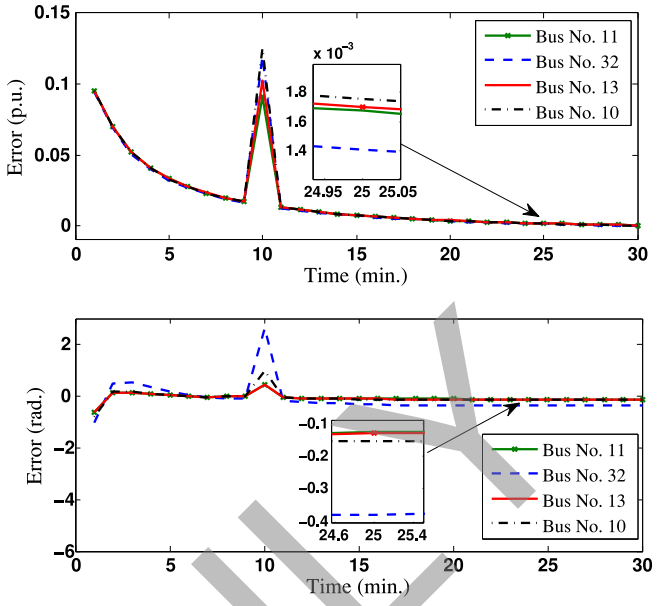

Fig. 12. Snapshot of estimation error for case 1 at bus numbers 10, 11, 13, and 32.

TABLE III
STATE ESTIMATION EXECUTION TIME

Case	T_{SE}^{CPU} LU	T_{SE}^{GPU} LU	T_{SE}^{CPU} PCG	T_{SE}^{GPU} PCG	S_p^{SE} LU	S_p^{SE} PCG
1	0.19s	0.13s	0.15s	0.06s	1.46	2.5
2	0.45s	0.24s	0.31s	0.12s	1.87	2.6
3	1.46s	0.59s	1.1s	0.33s	2.4	3.3
4	6.4s	2.05s	4.5s	1.2s	3.1	3.7
5	16.4s	3.4s	12.4s	2.6s	4.8	4.8
6	52.3s	7.2s	36.1s	5.4s	7.2	6.6
7	128s	9.8s	99.5s	8.5s	13	11.7
8	323s	15.91s	249.1s	12.9s	20.3	19.3

TABLE IV
DSE ERROR NORM (23) FOR DIFFERENT PERCENTAGE OF PMU INSTALLATION

Case	Normalized Error Norm in Voltage Magnitude			Normalized Error Norm in Phase Angle		
	$N_P = 10\%N_T$	$N_P = 20\%N_T$	$N_P = 40\%N_T$	$N_P = 10\%N_T$	$N_P = 20\%N_T$	$N_P = 40\%N_T$
1	0.004	0.0027	0.0015	0.59	0.37	0.23
2	0.0038	0.0028	0.0013	0.57	0.39	0.25
3	0.0036	0.0027	0.0015	0.55	0.32	0.29
4	0.0041	0.0029	0.0016	0.56	0.34	0.24
5	0.0033	0.0028	0.0017	0.58	0.36	0.27
6	0.0039	0.003	0.0017	0.59	0.37	0.28
7	0.0041	0.0026	0.0016	0.57	0.34	0.21
8	0.0036	0.0029	0.0015	0.56	0.38	0.28

to the next larger number. As it is shown the results are accurate for all of the case studies and the accuracy is increased as the percentage of PMU measurements increased.

D. Speed-Up and Complexity Analysis

As shown in Table II, the advantage of utilizing GPU for parallelization in MPDSE is significant when the size of the system is increased. One explanation is that for small size of data the communication overhead between the host and device

supersedes the execution time on the CPU. With growing system sizes, the CPU is barely able to handle the computation tasks in a reasonable time. With growing system sizes less time is spent on state estimation. The reason is that, as the size of the system grows the parallel portion of the program expands faster than the serial portion, while underscore GPUs advantages. The results of Table II are not unique due to the fact that the programming structure is one of the most important factors which affects the processing time. Therefore, a different programming style may lead to faster results; nevertheless, the speed-up shown in Table II would still be valid for increasing system sizes although with a slightly lower numerical value.

The portion of execution time related to actual DSE process (parameter identification, state prediction and state filtering) along with the actual state estimation process speed-up is reported in Table III. As the results show state estimation takes approximately 25%–30% of the total execution time of massive-thread DSE and between 35%–40% of the total execution time of the multithread DSE.

To analyze the algorithms complexity, their efficiencies are expressed as functions of the problem size. Considering S_c as the system scale, by curve fitting the closest growth rate function ($\mathrm{GR} = T_{\mathrm{Ex}}(S_c)$) for the CPU-based and the GPU-based algorithms are calculated as $S_c * \log_2(S_c)$ and $\sqrt{S_c} * \log_2(\sqrt{S_c})$, respectively. The growth rate functions for CPU ($\mathrm{GR}^{\mathrm{CPU}}$) and GPU ($\mathrm{GR}^{\mathrm{GPU}}$) are plotted in Fig. 9. As the size of the problem increases, the required time for execution increased proportional to these functions which proves that the speed-up will increase with growing system sizes. It can be seen that the CPU-based algorithm follows a higher order complexity compared to the GPU-based algorithm. The complexity reduction is the main advantage afforded by the GPU for MPDSE.

Distribution of threads, blocks and memory varies in different kernels. Typically, the number of thread per block is a constant number which was 128 in our case studies. The number of blocks per grid is different based on the problem size in each case study. In this paper, the maximum number of block per grid in each dimension was 16. GPU resource occupancy varied between 51% and 87% for LU decomposition and between 54% and 98% for PCG method.

V. Conclusion

The application of parallel processing for DSE is motivated by the desire for faster computation for real-time monitoring of the system behavior. The approach proposed in this paper investigates the process of accelerating the DSE for large-scale networks using both PMU and SCADA measurements. A data collation method is proposed where PMU devices are installed at a subset of buses, and the remaining measurements were extrapolated employing the exponential moving average method. A MPDSE process is formulated on the GPU. In addition, a multithreaded CPU implementation was developed for comparison. For a fair comparison exactly the same algorithm is used on both CPU and GPU. Numerical experiments in this paper, proved that successful parallelization utilizing iterative and direct linear solver results in a notable reduction in execution time; 15 times total speed-up for a 4992-bus power system is reported. The results verify the accuracy of the proposed GPU-based estimator under both normal and contingency conditions. If any optimization can happen to improve the CPU-only simulation, its implementation on the GPU would improve the GPU-based simulation as well. Future work will include detailed modeling of power system infrastructure to provide a higher accuracy and more detailed state estimation within the massively parallel framework.

References

[1] T. Van Cutsem and M. Ribbens-Pavella, "Critical survey of hierarchical methods for state estimation of electrical power systems," *IEEE Trans. Power App. Syst.,* vol. 102, no. 10, pp. 3415–3424, Oct. 1983.

[2] G. N. Korres and G. C. Contaxis, "Application of a reduced model to a distributed state estimator," in *Proc. 2000 IEEE Power Energy Soc. (PES),* vol. 2. Singapore, pp. 999–1004.

[3] A. Abur and A. Gómez-Expósito, *Power System State Estimation Theory and Implementation.* New York, NY, USA: Marcel Dekker, 2004, pp. 20–25.

[4] A. Gómez-Expósito and A. de la Villa Jaén, "Two-level state estimation with local measurement pre-processing," *IEEE Trans. Power Syst.,* vol. 24, no. 2, pp. 676–684, May 2009.

[5] C. Huanyuan, L. Xindong, S. Caiqi, and Y. Cheng, "Power system dynamic state estimation based on a new particle filter," in *Proc. IEEE Power Energy Soc. (PES),* San Diego, CA, USA, Jul. 2011, pp. 1–7.

[6] E. Ghahremani and I. Kamwa, "Dynamic state estimation in power system by applying the extended Kalman filter with unknown inputs to phasor measurements," in *Proc. Environ. Sci.,* vol. 11. Haikou, China, 2011, pp. 655–661.

[7] G. G. Rigatos, "A derivative-free Kalman filtering approach to state estimation-based control of nonlinear systems," *IEEE Trans. Ind. Electron.,* vol. 59, no. 10, pp. 3987–3997, Oct. 2012.

[8] S. Wang, W. Gao, and A. P. S. Meliopoulos, "An alternative method for power system dynamic state estimation based on unscented transform," *IEEE Trans. Power Syst.,* vol. 27, no. 2, pp. 942–950, May 2012.

[9] I. Dzafic, S. Henselmeyer, and H. T. Neisius, "High performance state estimation for smart grid distribution network operation," in *Proc. IEEE PES Conf. Innov. Smart Grid Technol.,* Hilton Anaheim, CA, USA, Jan. 2011, pp. 1–6.

[10] C. Gomez-Quiles, A. Gómez-Expósito, and A. de la Villa Jaén, "State estimation for smart distribution substations," *IEEE Trans. Smart Grid,* vol. 3, no. 2, pp. 986–995, Jun. 2012.

[11] L. Xie, D. H. Choi, S. Kar, and H. V. Poor, "Fully distributed state estimation for wide-area monitoring systems," *IEEE Trans. Smart Grid,* vol. 3, no. 3, pp. 1154–1169, Sep. 2012.

[12] L. Fan, Z. Miao, and Y. Wehbe, "Application of dynamic state and parameter estimation techniques on real-world data," *IEEE Trans. Smart Grid,* vol. 4, no. 2, pp. 1133–1141, Jun. 2013.

[13] K. C. Sou, H. Sandberg, and K. H. Johansson, "On the exact solution to a smart grid cyber-security analysis problem," *IEEE Trans. Smart Grid,* vol. 4, no. 2, pp. 856–865, Jun. 2013.

[14] O. Kosut, L. Jia, R. Thomas, and L. Tong, "Malicious data attacks on the smart grid," *IEEE Trans. Smart Grid,* vol. 2, no. 4, pp. 645–658, Dec. 2011.

[15] S. Zonouz *et al.*, "SCPSE: Security-oriented cyber-physical state estimation for power grid critical infrastructures," *IEEE Trans. Smart Grid,* vol. 3, no. 4, pp. 1790–1799, Dec. 2012.

[16] D. Blythe, "Rise of the graphics processor," *Proc. IEEE,* vol. 96, no. 5, pp. 761–778, May 2008.

[17] *CUDA C Programming Guide*, NVIDIA, Santa Clara, CA, USA, Feb. 2014.

[18] J. L. Gustafson, "Reevaluating Amdahl's law," *Commun. ACM,* vol. 31, no. 5, pp. 532–533, Jan. 1988.

[19] N. Husted, S. Myers, A. Shelat, and P. Grubbs, "GPU and CPU parallelization of honest-but-curious secure two-party computation," in *Proc. 2013 29th Comput. Security Appl.,* New Orleans, LA, USA, pp. 169–178.

[20] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units," in *Proc. IEEE Power Tech,* Lausanne, Switzerland, Jul. 2007, pp. 731–736.

[21] V. Jalili-Marandi and V. Dinavahi, "SIMD-based large-scale transient stability simulation on the graphics processing unit," *IEEE Trans. Power Syst.,* vol. 25, no. 3, pp. 1589–1599, Aug. 2010.

[22] C. Vilacha, J. C. Moreira, E. Miguez, and A. F. Otero, "Massive Jacobi power flow based on SIMD-processor," in *Proc. 10th Int. Conf. Environ. Elect. Eng.,* Rome, Italy, May 2011, pp. 1–4.
[23] Z. Li, V. D. Donde, J. C. Tournier, and F. Yang, "On limitations of traditional multi-core and potential of many-core processing architectures for sparse linear solvers used in large-scale power system applications," in *Proc. IEEE PES Power Energy Soc.,* San Diego, CA, USA, Jul. 2011, pp. 1–8.
[24] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," *IEEE Trans. Parallel Distrib. Syst.,* vol. 23, no. 7, pp. 1255–1266, Jul. 2012.
[25] Z. Zhou and V. Dinavahi, "Parallel massive-thread electromagnetic transient simulation on GPU," *IEEE Trans. Power Del.,* vol. 29, no. 3, pp. 1045–1053, Jun. 2014.
[26] A. G. Phadke, J. S. Thorp, and K. J. Karimi, "State estimation with phasor measurements," *IEEE Trans. Power Syst.,* vol. 1, no. 1, pp. 233–238, Feb. 1986.
[27] L. Zhao and A. Abur, "Multiarea state estimation using synchronized phasor measurements," *IEEE Trans. Power Syst.,* vol. 20, no. 2, pp. 611–617, May 2005.
[28] M. Zhou, V. A. Centeno, J. S. Thorp, and A. G. Phadke, "An alternative for including phasor measurements in state estimators," *IEEE Trans. Power Syst.,* vol. 21, no. 4, pp. 1930–1937, Nov. 2006.
[29] G. Valverde, S. Chakrabarti, E. Kyriakides, and V. Terzija, "A constrained formulation for hybrid state estimation," *IEEE Trans. Power Syst.,* vol. 26, no. 3, pp. 1102–1109, Aug. 2011.
[30] K. Das, J. Hazra, D. P. Seetharam, R. K. Reddi, and A. K. Sinha, "Real-time hybrid state estimation incorporating SCADA and PMU measurements," in *Proc. Innov. Smart Grid Technol.,* Berlin, Germany, Oct. 2012, pp. 1–8.
[31] M. Glavic and T. Van Cutsem, "Reconstructing and tracking network state from a limited number of synchrophasor measurements," *IEEE Trans. Power Syst.,* vol. 28, no. 2, pp. 1921–1929, May 2013.
[32] M. Gol and A. Abur, "Rapid tracking of bus voltages using synchrophasor assisted state estimator," in *Proc. IEEE Innov. Smart Grid Technol. Europe,* Lyngby, Denmark, Oct. 2013, pp. 1–5.
[33] Y. Chen, S. Jin, M. Rice, and Z. Huang, "Parallel state estimation assessment with practical data," in *Proc. 2013 Power Energy Soc. Gen. Meeting (PES),* Vancouver, BC, Canada, pp. 1–5.
[34] Y. Chakhchoukh, V. Vittal, and G. T. Heydt, "PMU based state estimation by integrating correlation," *IEEE Trans. Power Syst.,* vol. 29, no. 2, pp. 617–626, Mar. 2014.
[35] M. Asprou, E. Kyriakides, and M. Albu, "The effect of variable weights in a WLS state estimator considering instrument transformer uncertainties," *IEEE Trans. Instrum. Meas.,* vol. 63, no. 6, pp. 1484–1495, Jun. 2014.
[36] R. G. Brown, *Exponential Smoothing for Predicting Demand.* Cambridge, MA, USA: Arthur D. Little Inc., 1956.
[37] J. J. Shynk, *Probability, Random Variables, and Random Processes: Theory and Signal Processing Applications.* Hoboken, NJ, USA: Wiley, 2012.
[38] A. M. Leite da Silva, M. B. Do Coutto Filho, and J. F. de Queiroz, "State forecasting in electric power systems," *IEE Proc. Gener. Transmiss. Distrib.,* vol. 130, no. 5, pp. 237–244, Sep. 1983.
[39] W. Ding, J. Wang, and C. Rizos, "Improving adaptive Kalman estimation in GPS/INS integration," *J. Navigation,* vol. 60, no. 3, pp. 517–529, 2007.
[40] G. H. Golub and C. F. Van Loan, *Matrix Computations,* 3rd ed. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1996.
[41] *cuSPARSE Library,* NVIDIA Developer, Santa Clara, CA, USA, Feb. 2013.
[42] NVIDIA, "NVIDIA Tesla: A unified graphics and computing architecture," *NVIDIA CUDA C Programming Guide 4.0.,* Santa Clara, CA, USA, 2013.
[43] T. D. Han and T. S. Abdelrahman, "hiCUDA: High-level GPGPU programming," *IEEE Trans. Parallel Distrib. Syst.,* vol. 22, no. 1, pp. 78–90, Jan. 2011.
[44] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Comput. Sci. Eng.,* vol. 12, no. 3, pp. 66–72, May 2010.
[45] B. Chapman, G. Jost, and R. van der Pas, "Using openMP: Portable shared memory parallel programming," *IEEE Trans. Parallel Distrib. Syst.,* vol. 22, no. 1, pp. 78–90, Jan. 2011.

Hadis Karimipour (S'11) received the B.Sc. degree from the Ferdowsi University of Mashhad, Mashhhad, Iran, and the M.Sc. degree from the Shahrood University of Technology, Shahrood, Iran, in 2007 and 2009, respectively. She is currently pursuing the Ph.D. degree from the University of Alberta, Edmonton, AB, Canada, all in electrical engineering.

Her current research interests include large-scale power system state estimation, and parallel and distributed computing.

Venkata Dinavahi (SM'08) received the Ph.D. degree in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 2000.

He is currently a Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interests include real-time simulation of power systems, large-scale system simulation, and parallel and distributed computing.