Accelerated Parallel WLS State Estimation for Large-Scale Power Systems on GPU

Hadis Karimipour Venkata Dinavahi
Department of Electrical & Computer Engineering
University of Alberta
Edmonton, Alberta T6G 2V4, Canada.
Email: hadis@ualberta.ca, dinavahi@ualberta.ca

***Abstract*—Owing to the growing size and complexity of power networks, online monitoring of the power system state is a challenging computational problem. State estimation is paramount for reliable operation of large-scale power systems. Even with modern multi-core processors, the large number of computations in state estimation are still a burden on time and are highly memory intensive. In this paper the idea of using massively parallel graphic processing units (GPUs) for weighted least squares (WLS) based state estimation is introduced and executed. The GPU is especially designed for processing large data sets. A data-parallel implementation of the WLS method is proposed. The speed of the GPU-based state estimation for several test systems is compared with a sequential CPU-based program. The simulation results show a speed-up of 38 for a 4992-bus system.**

***Index Terms*—Data parallelism, Graphic processing units (GPUs), Parallel processing, Power system state estimation, Weighted least squares (WLS).**

I. Introduction

State estimation is a key energy management system (EMS) function, and is a vital part of power system operations in the control center [1]. The increasing size and complexity of modern power systems has made state estimation a slow and computationally expensive process. Therefore, many researchers have been motivated to investigate faster and numerically stable state estimation algorithms.

Among various state estimation methods, the weighted least squares (WLS) algorithm is the most commonly used method [2]. The WLS is fundamental for other algorithms, but for large-scale state estimation can be prohibitively slow and can be sensitive to measurement errors and noise. Advances in processor technologies and algorithmic enhancements such as parallel processing open up the possibility of faster state estimation for large-scale systems [3,4].

There exist two major processor architectures: the multi core computer processing unit (CPU) and the many core graphic processing unit (GPU). The CPU, which is the processor in most of the computers, is composed of only a few cores that can handle a few software threads at a time. In contrast, the GPU which is an energy-efficient processor on the market is composed of hundreds of cores that can simultaneously handle thousands of threads [5]. The massively parallel processing capability of the GPU has already started being exploited for power system applications such as power flow analysis [6-8] and transient stability simulation [9-11].

Utilizing the massively parallel architecture of the GPU, by assigning separate tasks to individual compute unified device architecture (CUDA) threads, computationally intensive sections of the state estimation program can be converted into a CUDA kernel (functional program which generates a large number of threads for data parallelism). Therefore, all the task can be off-loaded and executed in parallel utilizing thousands of threads, thereby accelerating the process of state estimation significantly.

In this paper the impact of the GPU thread parallelism on large-scale power system state estimation is demonstrated. To the best of our knowledge, such a work has not been reported in the literature. The WLS method is chosen as a candidate method for parallel implementation. There are several steps in each iteration of the WLS method including matrix-vector and matrix-matrix products which are computationally intensive. Using synchronized GPU threads, individual tasks in each iteration are parallelized. To achieve the maximum speed-up, every step is done on the GPU which significantly reduces the execution time. In addition, instead of matrix inversion which is one of the most computationally demanding component in state estimation, a parallel LU decomposition program is proposed as a solution for solving the linear equation which is the final step in state estimation. Comparison with the CPU-based sequential implementation shows speed-up's of 0.6 to 38 for different case studies. The results are verified in term of accuracy considering correlated and uncorrelated Gaussian noise in measurements.

II. WLS State Estimation Formulation

The weighted least squares method is a commonly used method for state estimation which tries to minimize the weighted sum of the squares of the residuals between the estimated and actual measurements [11]. Consider the measurement set vector $\boldsymbol{M}$ as:

$$\boldsymbol{M} = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\varepsilon}, \tag{1}$$

The bold notation refers to arrays; $\boldsymbol{M}$, $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{\varepsilon}$, are the vectors of measurements, nonlinear measurement functions, and measurement errors, respectively. For a system with n buses and m lines, there are $2m + 2n + 1$ elements in each

This work is supported by Natural Science and Engineering Research Council of Canada (NSERC).

978-1-4799-1255-1/13/$31.00 ©2013 IEEE

vector: $2m$ power flows, $2n$ power injections, and slack bus measurements. $\boldsymbol{x}$ is a vector of system states comprising of voltage magnitudes and phase angles. Since the phase angle in slack bus is considered 0, there are $2n-1$ states to be estimated. For simplicity, the error vector $\boldsymbol{\varepsilon}$ is assumed to be an uncorrelated Gaussian noise with zero mean. Substituting the first-order Taylor's expansion of $\boldsymbol{h}(\boldsymbol{x})$ around $\boldsymbol{x}_0$ in (1), we obtain:

$$\boldsymbol{M} - \boldsymbol{h}(\boldsymbol{x}_0) = \boldsymbol{H}\boldsymbol{\delta}(\boldsymbol{x}) + \boldsymbol{\varepsilon}, \tag{2}$$

where $\boldsymbol{H} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ is the $(2m+2n+1)\times(2n-1)$ Jacobian matrix and $\boldsymbol{\delta}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{x}_0$ is the $(2n-1)\times 1$ state mismatch vector. The objective function $\boldsymbol{J}(\boldsymbol{x})$ to be minimized by the WLS formulation can be expressed as:

$$\begin{aligned} J(x) &= \sum_{k=1}^{2m+2n+1} (M_k - h_k(x))^2 R_{kk}^{-1}, \\ &= [\boldsymbol{M} - \boldsymbol{h}(\boldsymbol{x})]^T \boldsymbol{R}^{-1} [\boldsymbol{M} - \boldsymbol{h}(\boldsymbol{x})], \end{aligned} \tag{3}$$

where $\boldsymbol{R}$ is the $(2m+2n+1)\times(2m+2n+1)$ covariance matrix. Index k refers to the k^{th} measurement. The following equation satisfies the first-order optimality condition at the minimum of $\boldsymbol{J}(\boldsymbol{x})$:

$$\boldsymbol{g}(\boldsymbol{x}) = \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{x}} = \boldsymbol{H}^T(\boldsymbol{x}) \boldsymbol{R}^{-1} [\boldsymbol{M} - \boldsymbol{h}(\boldsymbol{x})] = 0, \tag{4}$$

where $\boldsymbol{g}(\boldsymbol{x})$ is the $(2n-1)\times 1$ matrix of gradient of the objective function. Substituting the first-order Taylor's expansion of $\boldsymbol{g}(\boldsymbol{x})$ in (4), the following equation is solved iteratively to find the solution which minimizes $\boldsymbol{J}(\boldsymbol{x})$:

$$\boldsymbol{G}(\boldsymbol{x})\boldsymbol{\delta}(\boldsymbol{x}) = \boldsymbol{H}^T(\boldsymbol{x}) \boldsymbol{R}^{-1} [\boldsymbol{M} - \boldsymbol{h}(\boldsymbol{x})], \tag{5}$$

where $\boldsymbol{G}(\boldsymbol{x}) = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ is $(2n-1)\times(2n-1)$ gain matrix. The WLS state estimation algorithm given by (1)-(4) can be solved iteratively until convergence of $\boldsymbol{\delta}(\boldsymbol{x})$.

III. Graphics Processing Unit

GPU is specially designed to address mathematically expensive data-parallel problems. Since the GPU cannot work as a stand-alone processor, in most applications the CPU cooperates with the GPU.

A. GPU Architecture

The GPU is composed of hundreds of computational cores known as stream processors (SPs), unlike the CPU with a limited number of arithmetic cores. A multi-core CPU has 6 to 7 times larger cache than the GPU's cache system. On the other hand, the GPU has many more cores than the CPU. Owing to the fact that more transistors are devoted to data processing than caching and flow control, the GPU has significantly larger computational power compared to a multi-core CPU [12].

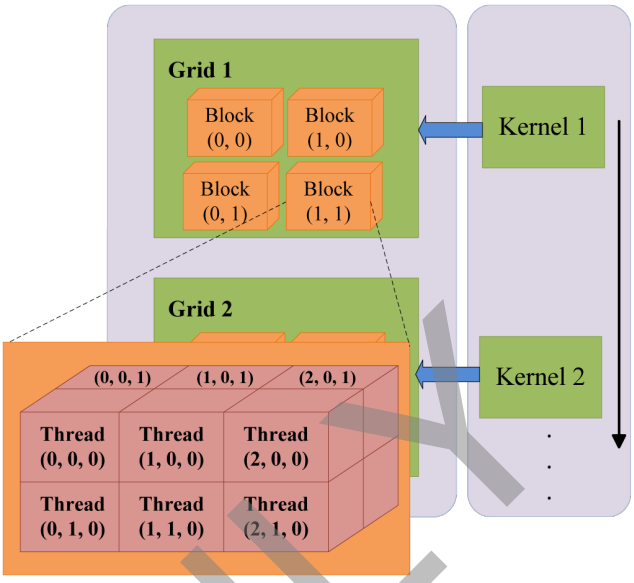

Fig. 1. CUDA thread organization for data-parallel processing [14].

B. CUDA Program Structure

A CUDA program consists of multiple phases that are executed on either the CPU (host) or the GPU (device). The sequential parts are implemented on the host, and intensive parallel phases are performed on the device. The GPU runs its own CUDA kernel independently under the CPU's control [13]. The execution starts with the host and moves to device after a kernel function is invoked. All the kernels have their own unique coordinates to distinguish themselves, and to identify the specific portion of the data to process. There is a two-level hierarchy in each thread named, *blockId* and *threadId*. The top level is a two dimensional array of blocks which is organized as a grid. All blocks in a grid have the same dimensions and share the same *blockId* values. Fig. 1 shows a grid of four blocks [14].

To ensure all threads in a block have completed a stage of their execution, *syncthreads()* is used to stop all threads until every thread in the block reached the same location.

C. Hardware Setup

The hardware used in this work is the Fermi GPU from NVIDIA. It has 512 cores which deliver up to 256 Gigaflops of double-precision peak performance. This device contains 16 streaming multiprocessor (SM), 32 cores, an instruction unit, and on-chip memory that come in three types: registers, shared memory, and cache [15]. The CPU is the Intel Xeon E5-2620, hexa-core with 2.0 GHz core clock and 32 GB memory, running 64-bit Windows 7 operating system.

IV. Massively Parallel WLS State Estimator

Although classical sequential state estimation methods result in a reliable and accurate estimation, their weakness is the long execution time which motivated the need for investigating options for acceleration in this work.

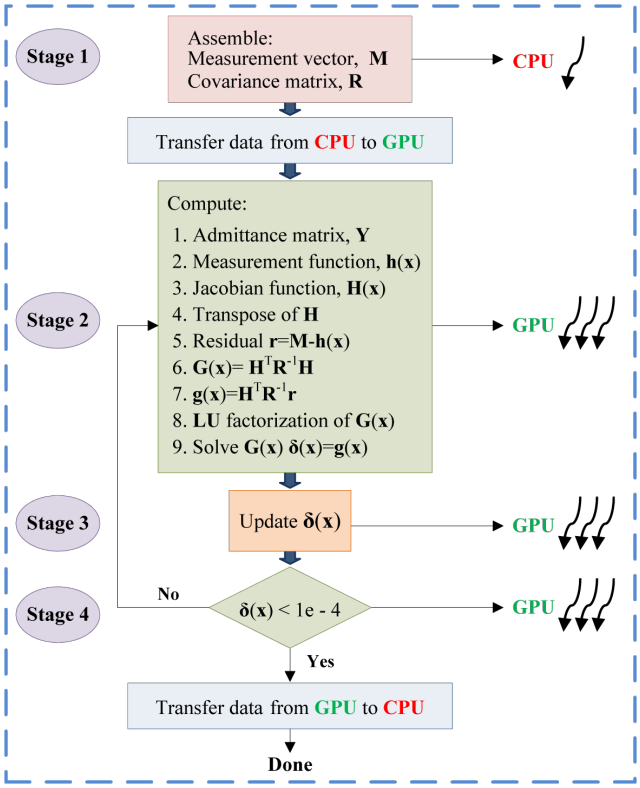

Fig. 2. GPU implementation of the WLS algorithm.

A. Parallel Kernel Structure

Generally, matrix-vector and matrix-matrix product is time consuming for large data-sets. In the WLS method, the computation of $\boldsymbol{H}^T(\boldsymbol{x})\boldsymbol{R}^{-1}[\boldsymbol{M}-\boldsymbol{h}(\boldsymbol{x})]$ and $\boldsymbol{H}^T(\boldsymbol{x})\boldsymbol{R}^{-1}\boldsymbol{H}$ can take a long time to complete even on CPU clusters. The matrix-vector multiplication to calculate $\boldsymbol{R}^{-1}[\boldsymbol{M}-\boldsymbol{h}(\boldsymbol{x})]$ takes about one order of magnitude less execution time than that of $\boldsymbol{H}^T(\boldsymbol{x})\boldsymbol{R}^{-1}[\boldsymbol{M}-\boldsymbol{h}(\boldsymbol{x})]$ or $\boldsymbol{H}^T(\boldsymbol{x})\boldsymbol{R}^{-1}\boldsymbol{H}$ but still can be a significant burden under emergency situations. Matrix-matrix and matrix-vector products contains several *for* loops in their implementation, which are the best candidates for parallelization utilizing GPU threads. Since all iterations of the loop can be executed in parallel, by assigning each iteration in a loop to individual CUDA threads, the task can be converted into a CUDA kernel [17].

In addition, solving $\boldsymbol{G}(\boldsymbol{x})\boldsymbol{\delta}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x})$ by inversion of $\boldsymbol{G}(\boldsymbol{x})$ is considerably expensive due to the sheer size of the inverted matrix. As an alternative method, dense LU decomposition is used in this work.

B. GPU Implementation

Although GPUs are well suited for large-scale data-parallel processing, writing efficient code for them is not without its difficulties. One of the most important bottleneck in parallel GPU programming is the overhead in data transfer between host and device and vise versa across a PCIe bus. The size and frequency of data transfers can create a bottleneck which significantly impacts the execution time of an algorithm. To achieve the most efficient result, all of these factors are considered in proposed GPU implementation.

Fig. 2 illustrates the implementation of WLS state estimator on the GPU. Initialization was done in Stage 1. After transferring the data from CPU to GPU, all the other steps were executed in the GPU. Stage 2 contains the main parts of the parallel kernel. All the vector products for the computation of admittance matrix $\boldsymbol{Y}$, measurement function $\boldsymbol{h}(\boldsymbol{x})$ and Jacobian function $\boldsymbol{H}(\boldsymbol{x})$ were done in parallel utilizing CUDA kernels. The transpose of $\boldsymbol{H}(\boldsymbol{x})$ or the computation of residual $\boldsymbol{r}$ are not intensive tasks like matrix or vector products, however for large-scale systems it can take significant execution time which is here reduced using parallel implementation. To compute the gain matrix $\boldsymbol{G}(\boldsymbol{x})$ and the gradient of the objective function $\boldsymbol{g}(\boldsymbol{x})$, the matrix-matrix multiplication $\boldsymbol{H}^T\boldsymbol{R}^{-1}$ was partitioned into a series of independent operations $\boldsymbol{H}_i^T\boldsymbol{R}_j^{-1}$ where $\boldsymbol{H}_i^T$ and $\boldsymbol{R}_j^{-1}$ refer to the i^{th} row and j^{th} column in $\boldsymbol{H}^T$ and $\boldsymbol{R}^{-1}$ matrices, respectively. The vector inner product of $\boldsymbol{H}_i^T\boldsymbol{R}_j^{-1}$ was defined as the sum of the elements of $\boldsymbol{H}_{i_a}^T\boldsymbol{R}_{j_b}^{-1}$ where $\boldsymbol{H}_{i_a}^T$ and $\boldsymbol{R}_{j_b}^{-1}$ are a^{th} and b^{th} element of the $\boldsymbol{H}_i^T$ and $\boldsymbol{R}_j^{-1}$, respectively. These independent operations are simultaneously executed by individual CUDA threads. In total $w \times z$ single threads were needed for product of $\boldsymbol{H}^T$ with w rows and $\boldsymbol{R}^{-1}$ with z columns. Using CUDA basic linear algebra subroutines (CUBLAS) library, $\boldsymbol{G}(\boldsymbol{x})$ was decomposed in parallel. Code for LU decomposition and for solving $\boldsymbol{\delta}(\boldsymbol{x})$ was prepared utilizing *cublasSscal()*, *cublasSswap()* and *cublasStrsv()* functions. After updating $\boldsymbol{\delta}(\boldsymbol{x})$ in Stage 3, convergence check was done in Stage 4.

V. Experimental Results and Discussion

In this section the results of the GPU implementation of the parallel WLS method are discussed for several case studies.

A. Test System

Large-scale systems were constructed to explore the efficiency of the GPU based state estimation simulations. Case 1 is the IEEE 39-bus system which was duplicated several times and interconnected via appropriate number of transmission lines to create large-scale systems [10]. Voltage magnitudes and angles of all buses are set to $1\angle 0$ p.u. for flat start. The inputs to the GPU-based WLS state estimator are the power-flow results from PSS/E corrupted with noise which are used as Pseudo-measurements. Therefore, to assess the accuracy of the state estimator, its output was compared with the original power-flow results from PSS/E.

B. Efficiency Evaluation

To demonstrate the efficiency of the GPU parallel code for WLS state estimation, the results are compared with the CPU simulations. Experiments are based on two separate simulation codes: one which ran sequentially on the CPU is in C++, and the other which is the integration of C++ and CUDA was prepared to be run on the GPU simulator. In most power system applications, and especially in state estimation, the

TABLE I
SUMMARY OF SIMULATION RESULTS

Case	No. of Buses	No. of Measurements	Jacobian matrix H(x)	Gain matrix G(x)	Single thread CPU	GPU Comm.	GPU Comp.	Speed-up
1	39	171	171×77	77×77	0.031s	0.017s	0.033s	0.6
2	78	347	347×155	155×155	0.18s	0.041s	0.069s	1.6
3	156	699	699×311	311×311	0.39s	0.08s	0.11s	2.05
4	312	1421	1421×623	623×623	2.7s	0.13s	0.38s	5.3
5	624	2865	2865×1247	1247×1247	16.5s	0.34s	1.56s	8.7
6	1248	5825	5825×2495	2495×2495	59.1s	0.86s	4.34s	11.4
7	2496	11553	11553×4991	4991×4991	195s	2.81s	10.19s	15
8	4992	23151	23151×9983	9983×9983	1577s	6.51s	34.99s	38

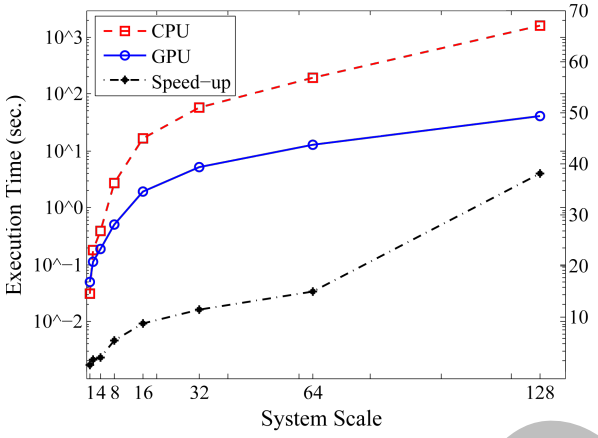

Fig. 3. Execution time and speed-up for various case studies.

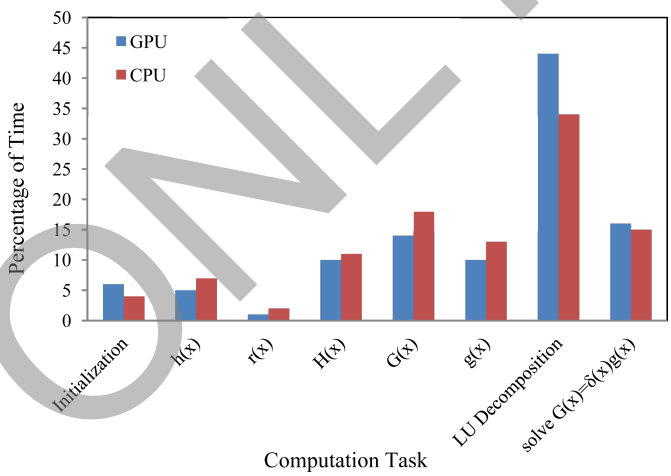

Fig. 4. Percentage of time used for various steps in Stage 2 (Fig. 2).

matrices are highly sparse. To reduce the computation burden, all matrices and vectors are stored in compressed row format. The simulations were done using the test data sets listed in Table 1, with a tolerance of 0.0001 for convergence of the estimated parameters.

Fig. 3 show the results of comparison between the CPU and the GPU simulators along with the speed-up of the parallel code. Although the proposed parallel simulator worked slower for small data sets, however, the advantages of utilizing the GPU in parallelization is significant when the size of the system increased. The reason is that for small size of data the communication overhead between host and device takes more time than the execution time in the CPU. With growing system size, the CPU is barely able to handle the computation tasks in a reasonable time. Unlike the CPU, the GPU is especially designed to handle large data processing. For Case 1 and Case 2, the speed of the GPU is not much more than that of the CPU which is what we expected to see. In contrast, for Case 3 and larger systems, the GPU's speed increases significantly, and for Case 8 the GPU was found to be 38 times faster than the CPU.

Since the programming structure is one of the most important factors which affects the processing time, it is probable to achieve faster results by multi-thread CPU programming. Nevertheless, the speed-up shown in Table 1 would still be valid for increasing system size although with a slightly lower numerical value.Fig. 4 illustrates the percentage of time taken by various steps in Stage 2 of Fig. 2 for Case 4. The results verify that for both the sequential CPU program and the data-parallel GPU program, the 3 most computationally demanding steps are the LU decomposition, computation of gain matrix $\boldsymbol{G}(\boldsymbol{x})$ and the solution of state mismatch vector $\boldsymbol{\delta}(\boldsymbol{x})$.

Based on Amdahl's law [16], even with many-core processors such as GPU, the maximum achievable speed-up utilizing parallel processing is limited by the time needed for the sequential part of the program. Gustafson's law which is a refined version of Amdahl's law argues that as the size of the problem increases, the inherently serial part of the program takes less portion in the overall problem. Since almost all the steps of our program is running on GPU, as the size of the system increases the parallel portion of GPU code expands faster than the serial portion. Based on Gustafson's law if X is a non-parallel fraction of a program, the highest possible speed-up using N processors is given as [17]:

$$S_p = N - X(N-1), \tag{6}$$

X can be calculated using the measured speed-up (S_m) on a specific number of processors (N_p) using

$$X = \frac{S_m - N_p}{(1 - N_p)S_m}. \tag{7}$$

In GPU programming the number of threads are changing in each step. Therefore, it is difficult to calculate the number of processors. For simplicity, it is assumed that given any number of GPU cores, all of them will be used to maximum capacity. The value of X for all of the case studies is calculated and the average value is used in Gustafson's law. In our case the average value of X is 0.39. Gustafson's law predicted a maximum speed-up of 312.7 using all 512 cores, which verifies the 38 speed-up in our case studies. Since the power of GPU will increase as the size of data increases, it is possible to achieve faster results up to predicted speed-up. Based on the results of Table 1, we believe that the maximum speed-up predicted by Gustafson's law is achievable for power systems with greater than 10000 buses.

VI. Accuracy Analysis of the GPU-based State Estimator

This section concentrates on the performance analysis of the GPU-based state estimator considering correlated and uncorrelated Guassian noise in measurement error.

A. Floating Point Precision

Due to the limitation in the number of bits for data storage and manipulation, it is necessary to round numbers when they cannot be adequately represented by a floating point number. It is possible that small inaccuracies due to rounding might result in an inaccurate final result. In addition, the outcome of the entire calculation is dependent on the order of basic arithmetic operations. The numerical operations on a GPU are performed in an environment that is much different from that on the CPU. Such aspects should be considered by the programmer to ensure accuracy of the simulation results. Depending on the efficiency of the written code, a GPU based program may produce results that differ from the CPU based code in an equivalent mathematical calculation [18].

B. Performance Analysis on GPU

There are two formats for representing floating point numbers: single precision and double precision format. Double precision numbers are double the size of single precision numbers and can present more range of numbers with more precision. In an application using floating point numbers the performance using a double precision format compared to the single precision format is slightly different. Since the proposed program is dealing with a lot more floating point numbers than integer numbers, the single precision format is used in programming. For applications that require accuracy greater than what can be given by single precision arithmetic, it is necessary that double precision numbers are used.

The results of Case 1 in Fig. 5 and Fig. 6 show that using single precision format the results are accurate enough. There are small differences in the result which are justifiable considering the fact that the order of blocks execution in each grid is undefined in kernel definition. Therefore, it leads to slightly different results if different blocks perform calculations on overlapping portions of data [18]. The estimation error for voltage magnitude and voltage angle in all of the case studies is less than 0.001 p.u. and 0.002 rad, respectively.

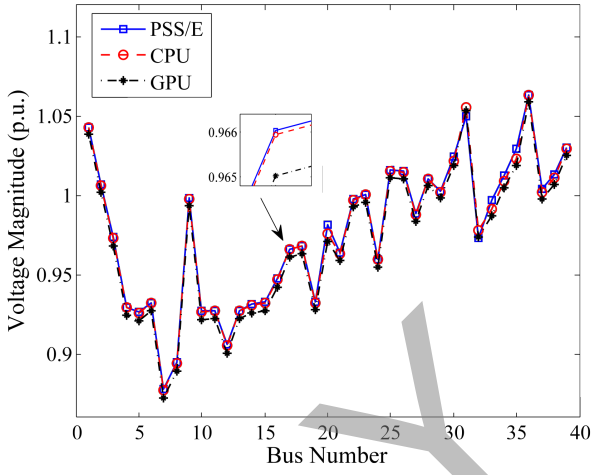

Fig. 5. Voltage magnitudes for Case 1.

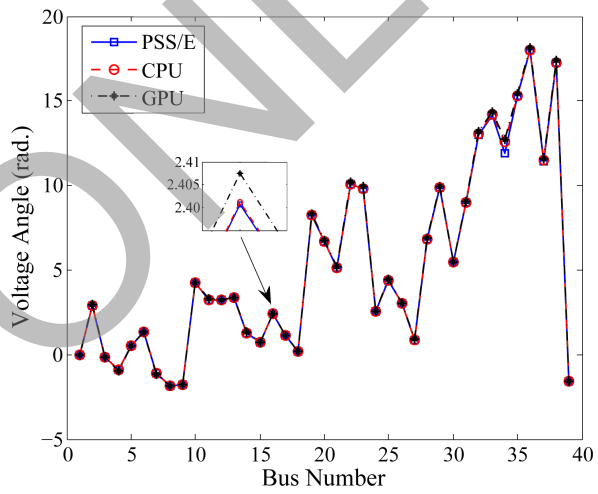

Fig. 6. Voltage angles for Case 1.

C. Effect of Measurement Errors on the GPU-based Estimator

The WLS method is based on the assumption that measurement errors are statistically distributed with zero mean. Apart from the technology used for measurements, it is important to examine whether there is bias in the measurements. Both uncorrelated Gaussian noise (*UGN*) and correlated Gaussian noise (*CGN*) which considers the affect of all other lines in the network in the measured value for a specific line, are considered. In the original WLS formulation, measurements error are uncorrelated and covariance matrix $\boldsymbol{R}$ is diagonal. With correlated Gaussian noise, $\boldsymbol{R}$ changes to a dense matrix. Therefore, all the matrix products of $\boldsymbol{R}$ contain larger values which leads to larger error in the final result.

As expected, the results in Fig. 7 and Fig. 8 show significant differences between the values of voltage magnitudes and

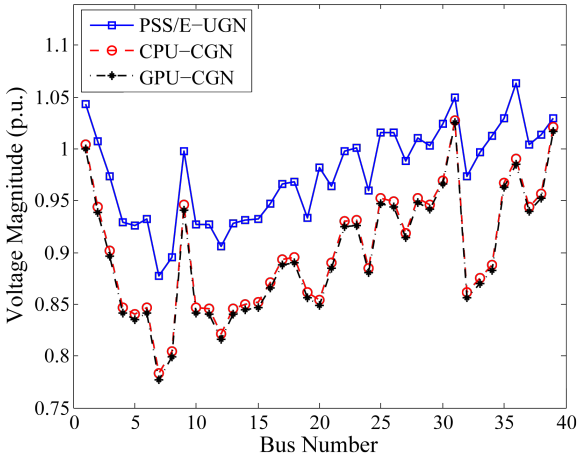

Fig. 7. Effect of correlated Gaussian noise on voltage magnitudes.

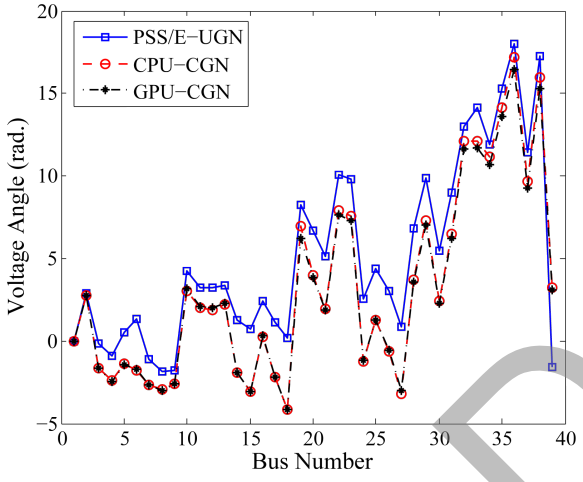

Fig. 8. Effect of correlated Gaussian noise on voltage angles.

angles using correlated and uncorrelated Gaussian noise. Both the GPU and the CPU programs have the same amount of error which verifies the accuracy of the GPU-based estimator in presence of noise. The results also prove that the WLS method is sensitive to noise and the effect of different type of noises should be considered in the formulation.

VII. Conclusion

This paper explored the process of accelerating the WLS-based state estimation for large-scale power systems using massively parallel graphic processing unit. Independent steps were conducted in parallel employing the GPU. Numerical experiments in this work proved that successful parallelization resulted in a notable reduction in execution time; 38 times faster state estimation for a 4992-bus power system is reported. In addition, the accuracy of the proposed method considering correlated and uncorrelated Gaussian noise is investigated. The results verify the accuracy of GPU-based estimator and sensitivity of the WLS method to noise.

References

[1] F. F. Wu, K. Moslehi, A. Bose, "Power system control centers: past, present, and future", *Proc. of the IEEE,* vol. 93, no. 11, pp. 1890-1908, Nov. 2005.
[2] A. Jain, N. R. Shivakumar, "Power system tracking and dynamic state estimation", *Proc. of IEEE PES General Meeting,* pp.1-8, Mar. 2009.
[3] D. J. Tylavsky, A. Bose, "Parallel processing in power systems computation", *IEEE Trans. on Power Systems,* vol. 7, no. 2, pp. 629-638, May 1992.
[4] D. M. Falciio, F. F. Wu, L. Murphy, "Parallel and distributed state estimation", *IEEE Trans. on Power Delivery,* vol. 3, no. 3, pp. 1099-1110, Jul. 1988.
[5] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, "NVIDIA Tesla: a unified graphics and computing architecture", *Proc. of IEEE Micro Conf.,* vol. 28, no. 2, pp. 39-55, Mar. 2008.
[6] A. Gopal, D. Niebur, S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units", *Proc. of Power Tech,IEEE Lausanne,* pp. 731-736, Jul. 2007.
[7] N. Garcia, "Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: a GPU-based approach", *Proc. of IEEE PES General Meeting,* pp. 1-4, Jul. 2010.
[8] C. Vilacha, J. C. Moreira, E. Miguez, A. F. Otero, "Massive Jacobi power flow based on SIMD-processor", *Proc. of 10th Int. Conf. on Envir. and Elect. Engin. (EEEIC),* pp. 1-4, May 2011.
[9] V. Jalili-Marandi, Z. Zhou, V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs", *IEEE Trans. on Parallel and Distributed Systems* , vol. 23, no. 7, pp. 1255-1266, Jul. 2012.
[10] V. Jalili-Marandi, V. Dinavahi, "SIMD-based large-scale transient stability simulation on the graphics processing unit", *IEEE Trans. on Power Systems,* vol. 25, no. 3, pp. 1589-1599, Aug. 2010.
[11] A. Abur, A. G. Exposito, "Power system state estimation theory and implementation", *Marcel Dekker, Inc.,* pp. 20-25, 2004.
[12] Z. Li, V. D. Donde, J. C. Tournier, 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", *Proc. of IEEE PES General Meeting,* pp. 1-8, Jul. 2011.
[13] T. D. Han, T. S. Abdelrahman, "hiCUDA: high-level GPGPU programming", *IEEE Trans. on Parallel and Distributed Systems,* vol. 22, no. 1, pp. 78-90, Jan. 2011.
[14] NVIDIA, "CUDA programming guide", Mar. 2012.
[15] NVIDIA, "NVIDIA's next neneration CUDA compute architecture: Fermi", June 2009.
[16] D. Blythe, "Rise of the graphics processor", *Proc. of IEEE,* vol. 96, no. 5, pp. 761-778, May 2008.
[17] J. L. Gustafson, "Reevaluating Amdahl's law", *Communications of the ACM,* vol. 31, no. 5, pp. 532-533, Jan. 1988.
[18] N. Whitehead, A. Fit-Florea, "Precision and performance: floating point and IEEE 754 compliance for NVIDIA GPUs", *NVIDIA Tech. rep,* April, 2012.

Hadis Karimipour is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada. Her research interests include parallel computing and state estimation for large-scale power systems.

Venkata Dinavahi is a professor at the University of Alberta, and his research interests include real-time simulation of electrical machines, power electronics and power systems, large-scale system simulation, and parallel and distributed computing.