International Journal of Smart Grid and Clean Energy

A GPU-Based Transient Stability Simulation Using Runge-Kutta Integration Algorithm

Zhijun Qin* and Yunhe Hou

Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China

Abstract

Graphics processing units (GPU) have been investigated to release the computational capability in various scientific applications. Recent research shows that prudential consideration needs to be given to take the advantages of GPUs while avoiding the deficiency. In this paper, the impact of GPU acceleration to implicit integrators and explicit integrators in transient stability is investigated. It is illustrated that implicit integrators, although more numerical stable than explicit ones, are not suitable for GPU acceleration. As a tradeoff between numerical stability and efficiency, an explicit 4th order Runge-Kutta integration algorithm is implemented for transient stability simulation based on hybrid CPU-GPU architecture. The differential equations of dynamic components are evaluated in GPU, while the linear network equations are solved in CPU using sparse direct solver. Simulation on IEEE 22-bus power system with 6 generators is reported to validate the feasibility of the proposed method.

Keywords: Transient stability simulation, Runge-Kutta algorithm, graphics processing units (GPU), parallel computing

1. Introduction

The transient stability simulation is a basic tool for security assessment for power grids. It involves the solution of a large number of differential and algebra equations (DAEs), which is one of the most computation-intensive tasks in power system analysis. Much research has been conducted to improve the efficiency of simulation to meet with the requirement of on-line security analysis. The integrated design of algorithms with specific hardware infrastructure is one of the important considerations in this area. The latest accomplishment in hardware accelerators provides new options to the implementation issue of transient stability simulation.

Programmable graphics processing units (GPUs) have evolved into a highly parallel, multithreaded and many-core processor to meet with the huge market demand for real-time, high-definition 3D graphics processing. As the programmability and performance of modern GPU increase, many application developers are looking to GPU to solve computationally intensive problems previously performed on general-purpose CPU.

The GPU-based framework for scientific computation consists of 3 levels: the basic linear algebra level, the basic algorithm framework level, and the application level.

For the basic linear algebra level, many high-efficient implemented libraries in CPU architecture as available, such as LAPACK [1], ATLAS [2], gotoBLAS [3], etc. Since solving dense linear systems of equations is a fundamental problem in scientific computing, it is critical to accelerate the solution of large-scale linear equations as fast as possible. The current efforts in GPU acceleration for dense linear algebra (DLA) is proposed in [4]. The method is based on hybridization techniques in the context of Cholesky, LU, and QR factorizations. In [5], an improved implementation of a triangular matrix inversion algorithm in multi-core CPU / dual-GPU systems is proposed.

* Manuscript received June 14, 2012; revised August 10, 2012.
Corresponding author. Tel.: +852-28578422; *E-mail address*: zjqin@eee.hku.hk.

As for the basic algorithm framework level, GPU is employed to accelerate the solution of linear optimization [6], nonlinear optimization [7], numerical integration [8], Monte Carlo simulation [9], etc. In [7], a parallelized implementation of generic algorithm to solve complex mix-integer nonlinear programming problems (MINLP) is proposed. It is shown that an acceleration of up to 20 times with double precision is achieved by GPU over CPU implementation.

In application level, applications of GPU in power system analysis and control are quite rare. It is natural to employ GPU in the visualization of power system data. A GPU-based contouring algorithm is proposed in [10] to implement efficient display of power system voltage levels. For GPU-based high performance computing in power system analysis, the research work has been starting. In [11], the authors use GPU with an calling interface provided by Microsoft C sharp (C#) to implement Jacobi algorithm for electromagnetic simulation, which has up to 10 times speedup compared with CPU-base implementation. In [12], an iterative biconjugate gradient method is employed for the solution of linear system in GPU, which accelerates the Newton method power flow calculation. Case study on IEEE 118 buses system shows that the proposed method is 2 times faster than CPU-based solution. A hybrid GPU-CPU simulator is designed in [13] to implement time-domain simulation using implicit trapezoidal rule. The maximum scale test case with 320 generators and 1248 buses has over 300X speedup over CPU solution.

However, it is not natural to transport CPU-based computation to GPU-based computation. On one hand, general-purpose applications, although have little to do with graphics, have to be programmed using graphics Application Programming Interfaces (APIs). Despite that various programming interfaces are developed for easy-development of GPU-based scientific applications, the legacy codes have to be rewritten conforming to GPU APIs. On the other hand, GPU is design for data-parallel computation and performs poor in dynamic memory allocation, complex flow control, etc. In other words, not all applications can be transported to GPU with efficiency improvement.

This paper addresses the challenges of transient stability simulation using GPU accelerations. To utilize the prominent data-parallel computational capability of GPU, explicit 4^{th} order Runge-Kutta integration is implemented and tested under hybrid CPU-GPU architecture. The proposed method uses the GPU computational capability in parallel evaluation of differential equations and the state-of-art sparse direct solvers to solve network equations in CPU architecture.

The remaining parts are organized as follows. Firstly, this paper investigates the impact of GPU in designing a transient stability simulation algorithm based on a hierarchical decision structure proposed in [14], which categories various numerical methods for time-domain simulation. The authors illustrate that GPU is suitable for explicit integration method. Secondly, a hybrid CPU-GPU platform is constructed. The hardware and software configuration of this platform is introduced. Thirdly, 4th order Runge-Kutta integration algorithm is implemented and tested.

2. Impact of GPU in Transient Stability Simulation

The transient stability simulation involves solving the differential-algebra equations (DAEs), which has the general form as:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \\ \boldsymbol{0} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}) \end{cases} \tag{1}$$

where $\boldsymbol{x}$ is the array of all state variables of dynamic components (generators, motors, excitation control, etc), $\boldsymbol{y}$ is the array of all algebra variables, mainly the buses voltages. The DAEs is converted into algebra equations by discretizing the differential equations.

The hierarchical decision structure on time-domain simulation algorithm design is proposed in [14], which consists of four layers as DAE construction strategy, integration method, nonlinear solver, and linear solver. The DAE construction strategy decides whether the differential equations and algebra

equations are solved alternately or simultaneously. Integration method determines how to discretize the differential equations. Nonlinear solver provides a framework to solve the algebra equations and discretized differential equations. Linear solver is responsible for solving the correction equations when seeking a solution for nonlinear equations.

In [15], two types of solutions for the overall system equations are summarized. One is partitioned solution with explicit integration, which converts (1) into the following form as:

$$\begin{cases} \boldsymbol{x}_{n+1} = \boldsymbol{F}(\boldsymbol{x}_n, \boldsymbol{y}_n) \\ \boldsymbol{0} = \boldsymbol{g}(\boldsymbol{x}_{n+1}, \boldsymbol{y}_{n+1}) \end{cases} \tag{2}$$

Where $\boldsymbol{x}_n$ is the state variable at the end of time interval n, $\boldsymbol{y}_n$ is the algebra variable at the end of time interval n. The discretized differential equations and algebra equations are solved independently. Since the solution of differential equations for $\boldsymbol{x}_{n+1}$ only needs the value $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ from previous time interval, it is substantially parallel for efficiency improvement. However, this solution may confront numerical instability, hence small time step is required.

The other solution is to convert (1) into (3), where the solution of differential equations for $\boldsymbol{x}_{n+1}$ needs to solve implicit equations related to $(\boldsymbol{x}_{n+1}, \boldsymbol{y}_{n+1}, \boldsymbol{x}_n, \boldsymbol{y}_n)$.

$$\begin{cases} \boldsymbol{x}_{n+1} = \boldsymbol{F}(\boldsymbol{x}_{n+1}, \boldsymbol{y}_{n+1}, \boldsymbol{x}_n, \boldsymbol{y}_n) \\ \boldsymbol{0} = \boldsymbol{g}(\boldsymbol{x}_{n+1}, \boldsymbol{y}_{n+1}) \end{cases} \tag{3}$$

The two parts of (3) can be solved simultaneously or alternately, depending on the property of the problems. The implementation is more complex and time-consuming than explicit solution, since it may have to solve large-scale nonlinear equations.

When considering acceleration of transient stability simulation based on GPU platform, we may consider: 1) whether a solution is suitable for GPU implementation; 2) how to implement parts of the solution with GPU; 3) whether there exists (or may exist in future) related products that could be utilized.

As illustrated by Fig. 1, the micro structure of GPU is design for data-parallel acceleration [16]. GPU instruction is executed by blocks of threads, each thread doing the same computation independently on different data elements. As a result, GPU performs poor in such areas as dynamic memory allocation, data cache, communication and collaboration among threads.

For example, in sparse-sparse matrix multiplication, if the result is also a sparse matrix, the amount of non-zero in each column of the result needs to be figured out during run-time. If sparse-sparse matrix multiplication is implemented in GPU, threads have to communicate to decide the sparse pattern of the result. Even worse, there is potential writing conflict among threads. In other words, the data-parallel mechanism fails, which deteriorates the efficiency of GPU implementation. Another example is sparse direct solver, which needs to figure out fill-in elements during factorization, where concurrent threads in GPU will not gain performance improvement unless the coefficient is a full matrix. Fortunately, sparse iterative solver, which utilizes the high-efficient implemented sparse matrix-vector multiplication, is applicable in solving large-scale network equations, as in [12]. In one word, sparse techniques are not fully compatible for the data-parallel mechanism of GPU, which affects the algorithm design for transient stability simulation.

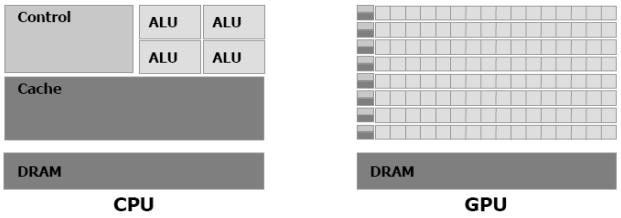

Fig. 1. Micro structure of CPU and GPU

The impact of GPU in the algorithm design of transient stability simulation is listed in Table 1. There exists many excellent sparse direct solvers under CPU architecture, such SuperLU [17], Umfpack [18], etc, which take the advantages of sparse techniques. As a comparison, iterative solvers, such as in [19], is implemented in GPU with data-parallel sparse matrix-vector multiplication. As for nonlinear solver, Newton-Raphson method is suitable for CPU, since it needs to factorize Jacobian matrix, which is highly sparse in transient stability simulation. When simultaneous implicit solution is implemented under GPU, the overall efficiency will not be improved due to the inadequate support for sparse techniques.

Table 1. Impact of GPU in algorithm design

	CPU compatible	GPU compatible
DAE construction	Simultaneous solution	Alternating solution
Integration method	Implicit integrator	Explicit integrator
Nonlinear solver	Newton-Raphson	Gauss-Seidel
Linear solver	LU factorization	Iterative solver

3. Transient Stability Simulation under Hybrid CPU-GPU Architecture

As mentioned in section II, the implicit integration solution is more numerical stable and time-consuming than explicit solution. High order integration algorithm or small integration step can be adopted to improve the numerical stability of explicit integration method. In this section, 4th order Runge-Kutta algorithm is adopted in transient stability simulation, which utilizes the parallel property of explicit integration algorithm and improves the numerical stability. What is more, the evaluation of differential equations is implemented under GPU, making full use of the intensive computational capability of GPU. Nevertheless, the algebra equations are solved using Newton-Raphson method with efficient sparse direct solver.

The general flow of transient stability can be described as follows. First, power flow analysis is performed to get the initial state of the power gird. Second, an iterative process is executed to determine all components' states and network' state at the end of each time interval of the interested time-horizon. At the beginning of each time interval, if a fault, or a switch event, or a control action happens, the linear complex network equation is solved to reflect the jump of bus voltages. During the time interval, the differential-algebra equations are solved to determine all variables' value at the end of the time interval, which are also the initial values of the next time interval. This process is repeated until the end of interested time-horizon. The flow chart is shown in Fig. 2.

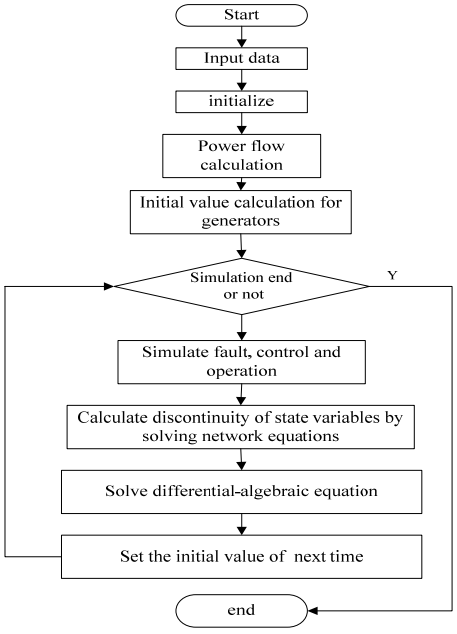

Fig. 2. General flow chart of transient stability simulation

To solve differential-algebra equations, 4th order Runge-Kutta algorithm is adopted using (4)~(12), where $\boldsymbol{f}(\cdot)\Delta t$ is the discretized form of $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$, and $\boldsymbol{g}(\cdot)$ represents the network equations, $\boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{K}_3, \boldsymbol{K}_4$ are the estimated change of state variable, $\tilde{\boldsymbol{x}}_{n+1}, \tilde{\boldsymbol{y}}_{n+1}$ are the state variable and algebra variable at the end of time interval *n+1*, respectively. When all loads are modeled as constant impedance branches, $\mathbf{g}(\cdot)$ becomes a linear complex equation with a sparse coefficient $\boldsymbol{Y}_n$, where $\boldsymbol{Y}_n$ is the admittance matrix during time interval *n*.

$$\boldsymbol{K}_1 = \boldsymbol{f}(\boldsymbol{x}_n, \boldsymbol{y}_n)\Delta t \tag{4}$$

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{x}_n + \frac{\boldsymbol{K}_1}{2}, \boldsymbol{y}_{k1}) \tag{5}$$

$$\boldsymbol{K}_2 = \boldsymbol{f}(\boldsymbol{x}_n + \frac{\boldsymbol{K}_1}{2}, \boldsymbol{y}_{k1})\Delta t \tag{6}$$

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{x}_n + \frac{\boldsymbol{K}_2}{2}, \boldsymbol{y}_{k2}) \tag{7}$$

$$\boldsymbol{K}_3 = \boldsymbol{f}(\boldsymbol{x}_n + \frac{\boldsymbol{K}_2}{2}, \boldsymbol{y}_{k2})\Delta t \tag{8}$$

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{x}_n + \boldsymbol{K}_3, \boldsymbol{y}_{k3}) \tag{9}$$

$$\boldsymbol{K}_4 = \boldsymbol{f}(\boldsymbol{x}_n + \boldsymbol{K}_3, \boldsymbol{y}_{k3})\Delta t \tag{10}$$

$$\tilde{\boldsymbol{x}}_{n+1} = \boldsymbol{x}_n + (\frac{\boldsymbol{K}_1}{6} + \frac{\boldsymbol{K}_2}{3} + \frac{\boldsymbol{K}_3}{3} + \frac{\boldsymbol{K}_4}{6}) \tag{11}$$

$$\mathbf{0} = \boldsymbol{g}(\tilde{\boldsymbol{x}}_{n+1}, \tilde{\boldsymbol{y}}_{n+1}) \tag{12}$$

Under a hybrid CPU-GPU architecture, (4), (6), (8), (10), and (11) are evaluated in GPU, while (5), (7), (9), and (12) are solved in CPU with sparse direct solver, as shown in Fig. 3.

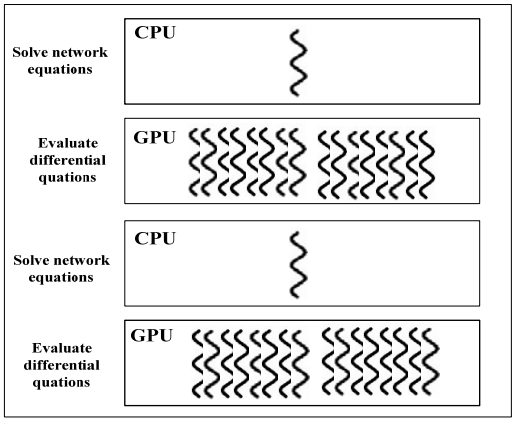

Fig. 3.Hybrid CPU-GPU architecture for transient stability simulation

We assume that all generators share the same dynamic model, and all controllers share the same model. In the implementation of GPU, thread blocks are issued to evaluate the differential equation, each thread evaluating one component of the same type. For example, one thread block is issued to simulate all generators, another thread block is issued to simulate all excitation controllers, etc.

When using NVIDIA's Compute unified device architecture (CUDA) programming model, various kernel functions are developed to simulate various types of dynamic components. A kernel function in CUDA is a subroutine executed in GPU by blocks of threads, each thread doing the same work on different data elements independently. The CUDA model provides simplicity in coding and transparent runtime scalability.

4. Case study

In this section, IEEE 22-bus system with 6 generators is used to demonstrate the feasibility of the proposed method. The size of the case is shown in Table 2. 4th order generator model is adopted in the simulation. The block diagrams of excitation controller and prime mover controller are shown in Fig. 4 and Fig. 5, respectively. All loads are modelled as constant impedance branches. As a result, the network equation becomes a linear complex one, which can be solved by LU factorization.

Table 2. Model size of case Study

Type	Number
Buses	22
Generators	6
Generators' variables	78
Excitation controllers' variables	66
Prime mover controllers' variables	66
Size of network equation	44

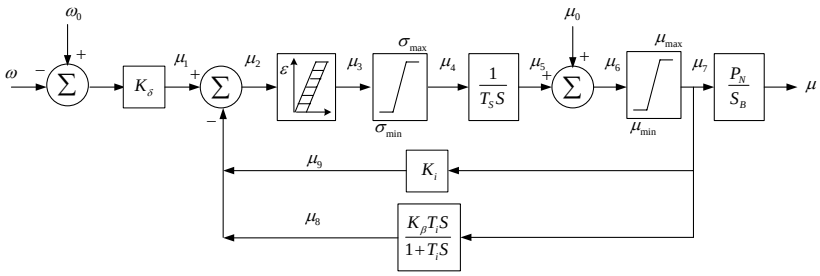

Fig. 4. Block diagram of prime mover controller

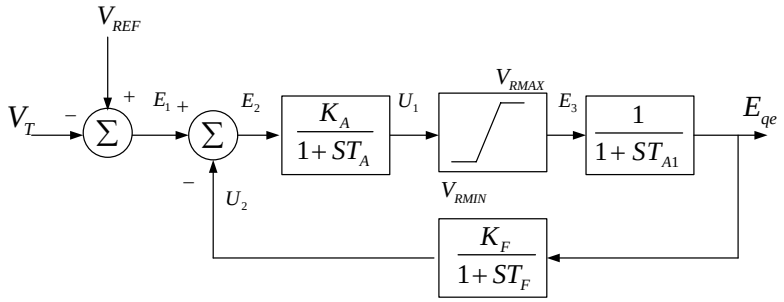

Fig. 5. Block diagram of excitation controller

Table 3. Fault setting and fault clearance setting

Fault/Clearance	Time
3-phase short circuit on head of line form bus 11 to bus 12	0.2 s
Trip breaks of line on both ends	0.4 s
3-phase short circuit cleared	0.5 s
Reclose breaks of line on both ends	0.9 s

The program is coded under Matlab v.2011b and Matlab GPU-addon Jacket v.2.0. The program runs on a personal computer with Intel i7 3.4GHz processor, one MSI N580GTX GPU card and 4 GB RAM.

The fault setting and fault clearance setting is shown in Table 3. The simulation duration is 0~6 seconds, 0.01 second as a step. The swing curves of all generators are shown in Fig. 6.

To test the scalability of the proposed method, the model is scaled by 10~100 times. The execution time of the program is shown in Fig. 7. It shows the execution time is scaled by 12 times when the system is scaled by 100 times.

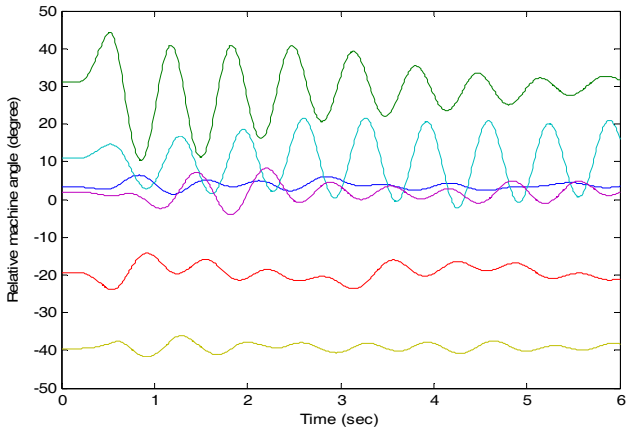

Fig. 6. Swing curves of all generators

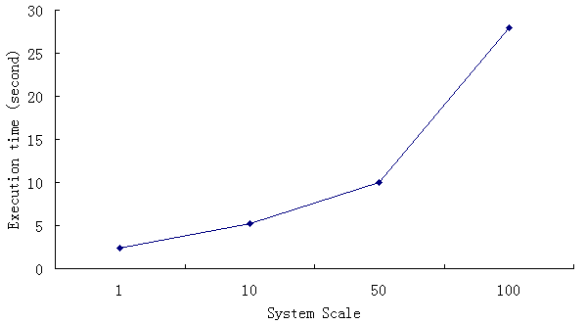

Fig. 7.Execution time of program

5. Conclusion

This paper reviews the algorithms for transient stability simulation. Although the implicit integration method for transient stability simulation has better numerical stability, it is more time-consuming than explicit ones due to the computational burden introduced by large-scale nonlinear equations. Moreover, implicit integrators depend on the sparse technology for efficiency improvement, which is not fully supported by GPU architecture.

As a trade-off between efficiency and numerical stability, this paper proposed a 4th order Runge-Kutta algorithm implemented under CPU-GPU architecture for transient stability simulation. The high order explicit integration algorithm mitigates the numerical instability problems. Further, the proposed method utilizes the data-parallel computational capability of GPU and the efficiency of the state-of-art sparse direct solvers under CPU. IEEE 22-bus system is used to validate the feasibility of the proposed method. It also shows the method is scalable for real size power grid simulations.

Acknowledgements

This work is partly supported by the Research Funding of EPRI (EP-P35571/C16133 , EP-P35424/C16059) , Research Grant Council, Hong Kong SAR（RGC-HKU 7124/10E, RGC-HKU 7124/11E）, National Basic Research Program(2012CB215102).

References

[1] LAPACK-Linear Algebra Package. [Online]. Available: http://www.netlib.org/lapack/.

[2] Demmel J, Dongarra J, Eijkhout V, Fuentes E, Petitet A, Vuduc R, Whaley RC, Yelick K. Self-adapting linear algebra

algorithms and software. *Proceedings of the IEEE*, 2005; 93(2):293-312.

[3] gotoBLAS, [Online]. Available: http://www.tacc.utexas.edu/tacc-projects/gotoblas2.

[4] Tomov S, Nath R, Ltaief H, Dongarra J. Dense linear algebra solvers for multicore with GPU accelerators. In: *Proc. of 2010 IEEE International Symposium on Parallel & Distributed Processing*, 2010:1-8.

[5] Ries F, De Marco T, Guerrieri R. Triangular matrix inversion on heterogeneous multicore systems. *IEEE Trans. on Parallel and Distribution Systems*, 2012; 23(1):177-184.

[6] Spampinato DG, Elster AC. Linear optimization on modern GPUs. In: *Proc. of IEEE International Symposium on Parallel & Distributed Processing*, 2009:1-8.

[7] Munawar A, Wahib M, Munetomo M, Akama K. Advanced generic algorithm to solve MINLP problems over GPU. In: *Proceedings of 2011 IEEE Congress on Evolutionary Computation*, 2011:318-325.

[8] Murray L. GPU acceleration of Runge-Kutta integrators. *IEEE Trans. on Parallel and Distribution Systems*, 2012; 23(1):94-101.

[9] Weber R, Gothandaraman A, Hinde RJ, Peterson GD. Comparing hardware accelerators in scientific applications: a case study. *IEEE Trans. on Parallel and Distribution Systems*, 2011; 22(1):58-68.

[10] Euzebe TJ, Overbye TJ. Contouring for power system using graphical processing units. In: *Proc. of the 41st Annual International Conference on System Science*, 2008:168.

[11] Woolsey M., Hutchcraft WE, Gordon RK, High-level programming of graphics hardware to increase performance of electromagnetic simulation. In: *Proc. of 2007 IEEE Antennas and Propagation Society International Symposium*, 2007:5925-5928.

[12] Garcia N. Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: a GPU-based approach. In: *Proc. of 2010 IEEE Power and Energy Society General Meeting*, 2010:1-4.

[13] Jalili-Marandi V, Dinavahi V. SIMD-based large-scale transient stability simulation on the graphics processing unit. *IEEE Trans.* on *Power Systems*, 2010; 25(3):1589-1599.

[14] Fu C, McCalley JD, Tong JZ. A numerical solver design for extended-term time-domain simulation. *IEEE Trans. on Power Systems*, 2012

[15] P. Kundur, *Power System Stability and Control*, New York: McGraw-Hill; 1994.

[16] http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_ Guide.pdf

[17] http://www.cs.berkeley.edu/~demmel/SuperLU.html.

[18] http://www.cise.ufl.edu/research/sparse/umfpack.

[19] Wang M, Klie H, Parashar M, Sudan H. Solving sparse linear systems on NVIDIA Tesla GPUs. In: *Proc. of Computational Science—ICCS 2009*, 2009:864–873.