Large-Scale Transient Stability Simulation on Graphics Processing Units

Vahid Jalili-Marandi, *Student Member, IEEE,* Venkata Dinavahi, *Senior Member, IEEE*

***Abstract*— Graphics processing units (GPUs) have recently attracted a lot of interest in several fields struggling with massively large computation tasks. The application of a GPU for fast and accurate transient stability simulation of the large-scale power systems is presented in this paper. The computationally intensive parts of the simulation were offloaded to the GPU to co-operate with the CPU. As such, a hybrid GPU-CPU simulator is configured. The accuracy of the proposed simulation approach has been validated by using the PSS/E software. The computation time of the simulation performed by co-processing of GPU-CPU has been compared with that of the CPU-only simulation. Several test cases have been used to demonstrate the significant acceleration of the GPU-CPU simulation. A speed-up of 345 is reported for a 320 generator and 1248 bus power system.**

***Index Terms*— Graphics processors, Parallel programming, Power system transient stability.**

I. INTRODUCTION

TRANSIENT stability study is important for planning, operation, and control of modern electrical power systems. Transient stability simulation of a realistic multi-machine power system requires the solution of a large set of non-linear differential-algebraic equations in the time-domain that asks for significant computational resources. The standard method for transient stability simulation [1] involves three steps: (1) discretization of the differential equations, (2) iterative solution of the non-linear algebraic equations, and (3) solution of the linear algebraic equations. Several approaches have been developed in the past to perform this simulation faster on parallel hardware [2]. The computations performed in these implementations, however, were still sequential.

In this paper, we demonstrate how the application of a Graphics Processing Unit (GPU) can yield significant speed-up in large-scale transient stability simulation using multi-threaded parallel programming. The GPU was originally developed to meet the needs for fast graphics in the gaming and animation industries. The need for life-like rendering of characters in these areas led to further advancement in its processing capabilities and programmability. Taking advantage of its massively parallel hardware architecture, starting in the early 2000's, the applications of GPU mushroomed to include intensive general purpose computations such as those in molecular biology, image and video processing, n-body simulations, large-scale database management, and financial services [3].

Financial support from the Natural Science and Engineering Research Council of Canada (NSERC) is gratefully acknowledged.

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta T6G 2V4, Canada. Email: [v_jalili, dinavahi]@ece.ualberta.ca

978-1-4244-4241-6/09/$25.00 ©2009 IEEE

Although there have been reported applications [4], [5] where all the computation tasks are done in a single GPU or a cluster of GPUs, the focus of this paper is a hybrid GPU-CPU simulator where GPU and CPU co-operate to perform a simulation. In these cases GPU acts as a co-processor beside the CPU to do the intensive computation tasks, whereas CPU is mostly responsible for the flow control of the simulation or updating the required variables. The paper is organized as follows: Section II gives an overview of the hardware structure of the GPU, and the parallel computing philosophy. In Section III mathematic formulation of the transient stability study in power systems and the model used in this work are explained. Section IV presents the GPU-CPU co-processing results for transient stability simulation and related discussions. The conclusion of this paper appears in Section V.

II. GRAPHICS PROCESSING UNIT

A. Hardware structure and specifications

Fig. 1 illustrates the Tesla GPU architecture introduced by NVIDIA [7], and also its connection to a PC. The GPU runs its own specified instructions independently from the CPU but it is controlled by the CPU. A *Thread* is the computing element in the GPU. When a GPU instruction is invoked, blocks of threads (with the maximum size of 512 threads per block) are defined to assign one thread to each data element. All threads in one block run the same instruction on one streaming multiprocessor (SM) or thread processing arrays (TPAs), which gives the GPU a SIMD (Single-Instruction, Multiple-Data) architecture. Each SM includes 8 stream processor (SP) cores, an instruction unit, and on-chip memory that comes in three types: registers, shared memory, and cache (constant and texture). Threads in each block have access to the shared memory in the SM, as well as to a global memory in the GPU. Unlike a CPU, the inside structure of a GPU is developed in such a way that more transistors are devoted to data processing rather than data caching and flow control. When a SM is assigned to execute one or more thread blocks, the instruction unit splits and creates groups of parallel threads called *warps*. The threads in one warp are managed and processed concurrently. Depending on the GPU model, 2 or 3 SM's can be clustered to build a thread processing cluster (TPC). Threads are assigned by the thread scheduler which talks directly to each SM through a dedicated instruction unit which in turn assigns the tasks to the eight thread (or stream) processors. Moreover two special function units (SFUs) have been included in each SM to execute transcendental calculations (e.g. sin, cosine), attribute interpolation and for executing floating point instructions.

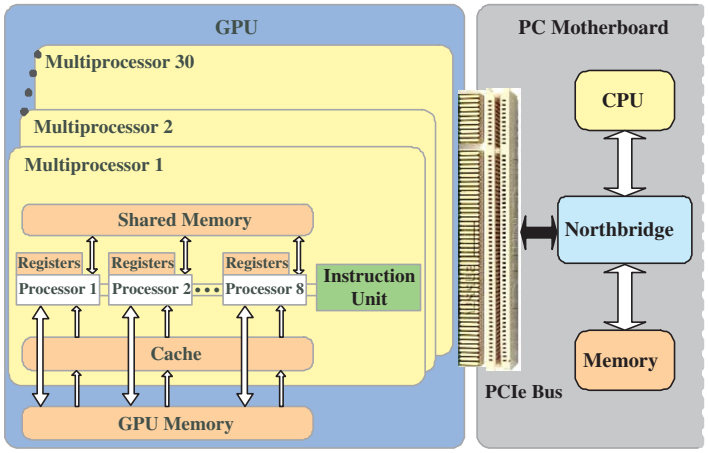

Fig. 1. Hardware architecture of GPU mounted on the PC motherboard.

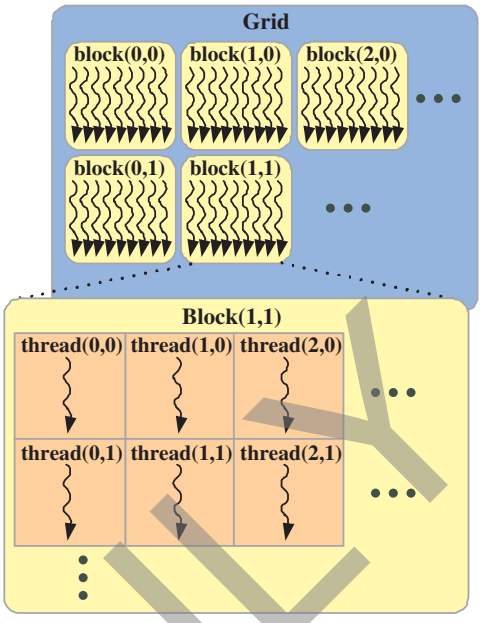

Fig. 2. Hierarchy of computing structure in a GPU (figure modified from [7]).

In this work, the GPU is an NVIDIA GeForce GTX 280 (30 SMs consisting of 240 SPs clocked at 1296MHz with 1GB of GPU device memory with a bandwidth of 141.7GB/s). Each SM in the GPU can process 32 active warps at a time with 32 threads per warp, resulting in concurrent processing of 1024 threads. Supporting both the single-precision and double-precision floating point numbers, the theoretical peak performance of GTX 280 is 933 GFLOPS. Fig. 3 shows the detailed architecture of the GTX 280. The GPU is plugged into the motherboard of a 2.5GHz quad-core AMD Phenom CPU supported by 4GB of RAM. The GPU and CPU communicate via the PCIe 2.0x16 bus that supports up to 8GB/s transfer rate.

B. SIMD-based parallel programming

In the SIMD-based architecture each data element is mapped to one processor core and is executed independently. For example two vectors can be added sequentially on a CPU, as follows:

$for(i = 0; i < 100; i++)$
$\quad a[i] \leftarrow b[i] + c[i]$

This operation can be performed on a SIMD device in parallel, if each element of arrays b and c are added simultaneously. Therefore, all 100 iterations in the above for loop can be executed concurrently, as follows:

$if(index < 100)$
$\quad a[index] \leftarrow b[index] + c[index]$

where $index$ is the ID of the threads assigned to elements of two vectors. Designed for general purpose GPU computing applications, CUDA (Compute Unified Device Architecture) is the environment for NVIDIA's GPU programming [7]. CUDA is an extension of the C language, and it allows a programmer to define functions called *kernels*. Whenever a kernel is invoked it is run N times in parallel in N separate threads on a CUDA-enabled *device* (i.e. GPU) that operates as a co-processor for the *host* (i.e. CPU), which runs the C program. A kernel is executed on a *grid* consisting of several blocks with equal numbers of threads. As illustrated in Fig. 2, each block within a grid, and each thread within a block are identified and accessible by individual indices. The general syntax to invoke a kernel, called *testKERN*, is as follows:

testKERN<<<dimGrid, dimBlock>>>(A, B);

where arguments of the kernel are enclosed in parentheses, and the dimensions of the grid and its blocks are specified by *dimGrid* and *dimBlock* respectively. Moreover on top of CUDA a library of the basic linear algebraic functions is provided that allows the integration with C++ code. By using this library, called CUBLAS, portions of a sequential C++ program can be executed in parallel on the GPU, Fig. 4, while other parts of the code , which are in C++ language, are executed sequentially on the CPU [8]. Therefore, in an application with highly intensive computations, the onerous computation tasks can be offloaded to the GPU, and performed faster in parallel, whereas the flow control of the program, required initial calculations, or the updating and saving of the variables can be done by the CPU. This co-processing configures a hybrid GPU-CPU simulator.

III. TRANSIENT STABILITY SIMULATION OF LARGE-SCALE SYSTEMS

A widely used method for detailed modeling of synchronous generator for transient stability simulation is to use Park's equations with an individual dq reference frame fixed on the generator's field winding [9]. The network, including transmission lines and loads, is modeled using algebraic equations in a common dq reference frame. Representation of AVR and PSS increases the number of differential equations and hence the complexity of the model. The validity of the dynamic response in a network with a lot of interconnections and in a time frame of few seconds highly depends on the accuracy of the generator model and other components which can have effects on the dynamics of the system. The general form of differential-algebraic equations which describe the

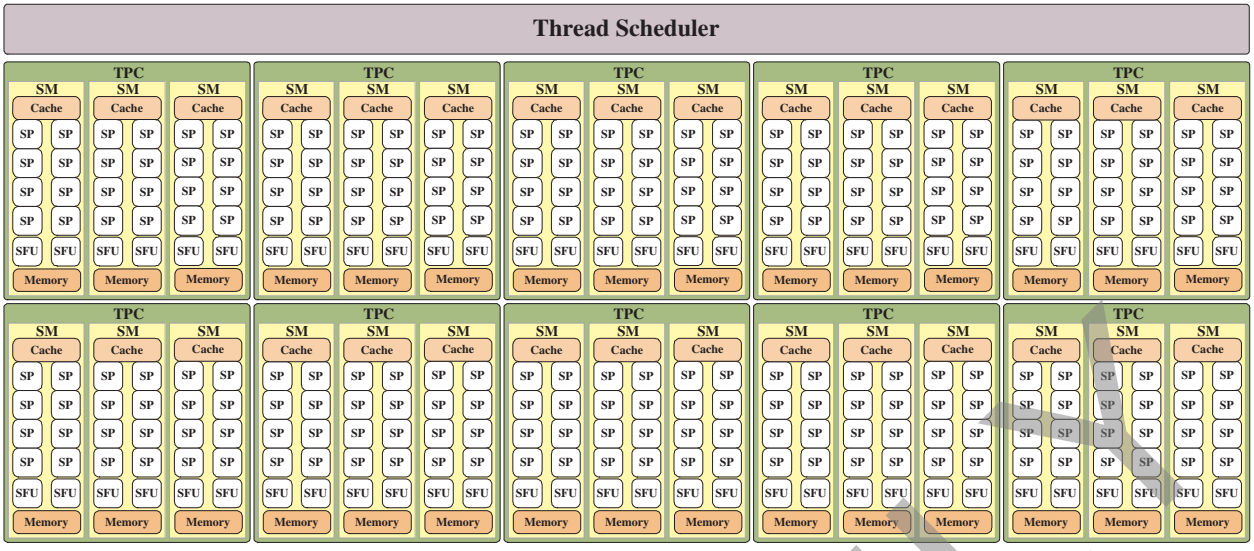

Fig. 3. The GTX 280 GPU architecture: it consists of 10 TPCs, 3 SMs/TPC, and a total of 240 SPs; TPC: thread processing cluster, SM: streaming multiprocessor, SP: stream processor (figure modified from [6]).

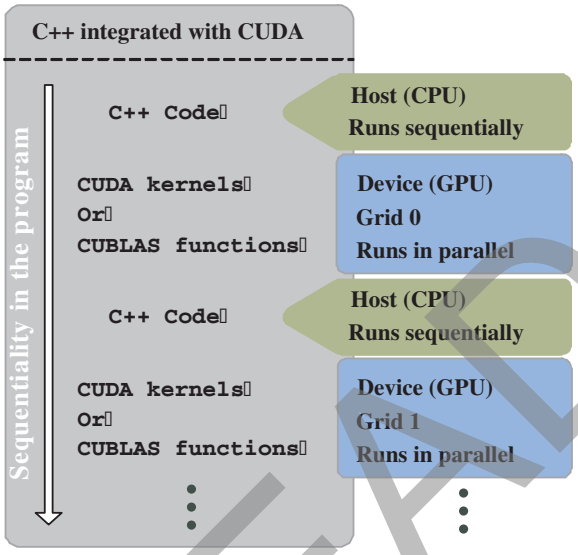

Fig. 4. Co-operation of the host and device to execute a C++ code integrated with CUDA.

dynamics of a multi-machine power system assuming single-phase, positive sequence, and fundamental frequency behavior, is given as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{V}, t), \quad (1)$$

$$\mathbf{0} = \mathbf{g}(\mathbf{x}, \mathbf{V}, t), \quad (2)$$

$$\mathbf{x}(t_0) = \mathbf{x_0}, \quad (3)$$

where $\mathbf{x}$ is the vector of state variables, $\mathbf{x_0}$ is the initial values of state variables, and $\mathbf{V}$ is the vector of bus voltages. In this work the detailed model (Appendix A) of synchronous generator including AVR and PSS is used. Therefore, each generating unit consists of 9 state variables. In a power network with m synchronous generators and n buses, $\mathbf{x}$ is a $9m \times 1$ vector and $\mathbf{V}$ is a $2n \times 1$ vector.

The numerical method to perform transient stability study is finding the solution of this set of nonlinear differential-algebraic equations (i.e. (1) and (2)) over the interested period of time. The standard approach consists of three major steps: discretization of non-linear differential equations, linearization of non-linear algebraic equations, and solution of linear algebraic equations. Discretizing (1) results in a new set of non-linear algebraic equations. In this work we used the trapezoidal rule as the implicit integration method to discretize the differential equations as follows:

$$0 = \mathbf{x} - \frac{h}{2}\left[\mathbf{f}(\mathbf{x}, \mathbf{V}, t) + f(\mathbf{x}, \mathbf{V}, t-h)\right], \quad (4)$$

where h is the integration time-step. (2) and (4) can be linearized by the Newton-Raphson method (for the j^{th} iteration) as:

$$J(\mathbf{z}^{j-1}) \cdot \Delta\mathbf{z} = -\mathbf{F}(\mathbf{z}^{j-1}), \quad (5)$$

where J is the Jacobian matrix, $\mathbf{z} = [\mathbf{x}, \mathbf{V}]$, $\Delta\mathbf{z} = \mathbf{z}^j - \mathbf{z}^{j-1}$, and $\mathbf{F}$ is the vector of nonlinear function evaluations. (5) is a set of linear algebraic equations that can be solved with Gaussian Elimination and back substitution method.

Benchmarking revealed that a majority of execution time in a transient stability simulation is spent for steps (2) and (3), i.e. the nonlinear iterative solution using Newton-Raphson, and the linear algebraic equation solution. By exploiting a hybrid GPU-CPU simulator, as defined in the previous section, these two steps of the simulation were off-loaded to the GPU to be processed in parallel, whereas the remaining tasks such as discretizing, updating, and computation of intermediate variables were executed sequentially on the CPU.

IV. Implementation Results

This section verifies the accuracy and efficiency of the hybrid GPU-CPU co-processing for transient stability simulation of large-scale systems.

A. Simulation accuracy evaluation

In this part we show the results of applying the simulation code, prepared by integrating C++ and CUBLAS library, on a case study. The sincerity of our simulation code will be validated by using the PTI's PSS/E software program. The case study used in this section is the IEEE 39 bus New England test system whose one-line diagram is plotted in Fig. 5. All generator models are detailed and equipped with AVR and PSS described in Appendix A. The complete system can be described by 87 non-linear differential and 20 algebraic equations. Several fault locations have been tested and the results were compared with those of PSS/E. In all cases results from GPU-CPU co-processing method match very well. This paper presents a sample of these results. A three-phase fault happens at Bus 21, at t=1s and it is cleared after 100ms. $Gen10$ is the reference generator and the relative machine angles are shown in Fig. 6 and Fig. 7. To show the comparison the similar results found from PSS/E are superimposed on top of these two figures. As can be seen our transient stability code is completely stable during the steady-state of the system, i.e. t<1s. During the transient state and also after the fault is cleared, our program results closely follow the results from PSS/E. The maximum discrepancy between generator angles resulted in GPU-CPU co-processing simulation and PSS/E was found to be 1.46%, based on (6):

$$\varepsilon_\delta = \frac{max|\delta_{PSS/E} - \delta_{GC}|}{\delta_{PSS/E}}, \tag{6}$$

where $\delta_{PSS/E}$ and δ_{GC} were defined as the relative machine angles from PSS/E and GPU-CPU co-processing simulation, respectively.

B. Computational efficiency evaluation

To investigate the efficiency of the proposed hybrid GPU-CPU simulator for transient stability study of large-scale power systems, we show the comparative results in this section. Several systems have been used for this evaluation which their specifications are listed in Table I. The system with $Scale$ 1 is the IEEE's New England test system, illustrated in Fig. 5 and verified in the previous section. The $Scale$ 1 system was duplicated several times to create systems of larger scales. Therefore we obtained test systems of 78, 156, 312, 624, and 1248 buses. In these systems we used a flat start, i.e. voltage and angle of all buses set to $1.0\angle 0°$p.u., and modeled them in the PSS/E software to find the load flow results. These results were then fed into the prepared simulation codes.

Two separate simulation codes were prepared: one code is purely in C++ to be run sequentially on the CPU, and the other is C++ integrated with CUDA to be run on the hybrid simulator (this code was used and validated in the previous section). The GPU-CPU co-processing was compared with a CPU-only processing. In Table I the columns indicated by CPU and GPU-CPU list the computation time to simulate a duration of 1580ms with a time-step of 10ms for all systems. GPU is truly advantageous for parallel computing on a large set of input data. For small size of data, the communication overhead and memory latency in GPU are not insignificant in

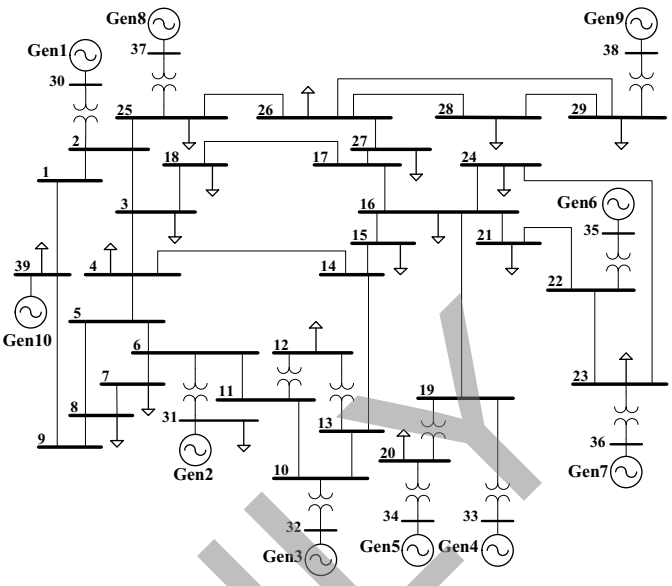

Fig. 5. One-line diagram for the IEEE 39 bus power system ($Scale$ 1 system).

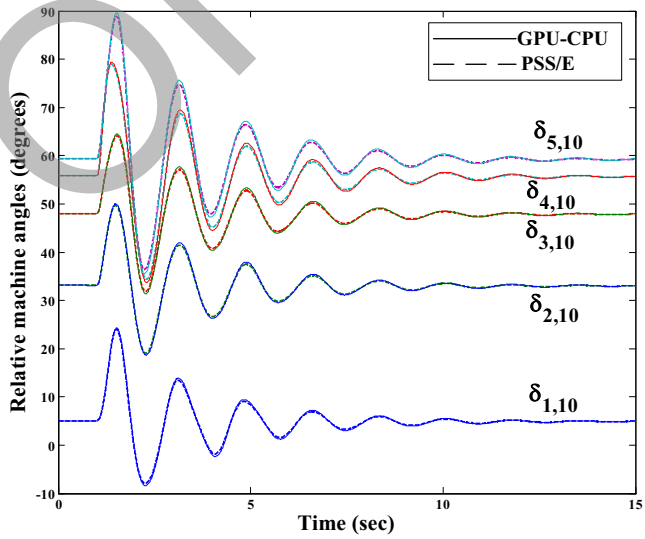

Fig. 6. Comparison of relative machine angles collected from hybrid simulator and PSS/E simulation for IEEE 39 bus test system: $\delta_{i,10} = \delta_i - \delta_{10}$; $i = 1 \ldots 5$.

comparison with the computation time. As such, we do not expect better performance for $Scale$ 1 and $Scale$ 2 systems. When the size of system increases, however, the latency is dwarfed by the computation time, and GPU shows a very high speed-up. For example, for $Scale$ 32, GPU takes 1m 44.4s for simulation, whereas the CPU needs 10hrs. The effect of SIMD-based parallel computing on the acceleration of simulation can be observed from the speed-up graph in Fig. 8. Speed-up is the ratio of the CPU and GPU-CPU computation times. We can see that for $Scale$ 32 GPU-CPU co-processing is more than 340 times faster than CPU-only processing.

The maximum achievable speed-up by parallelizing some portions of a sequential algorithm can be verified by Amdahl's law, which states that if f $(0 \leqslant f \leqslant 1)$ is the fraction of the

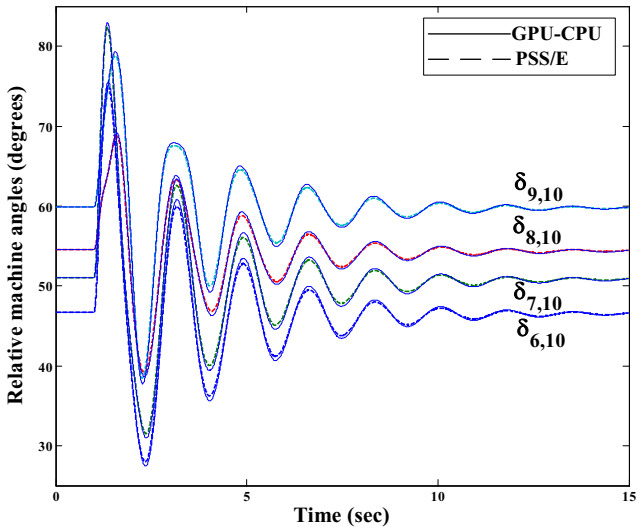

Fig. 7. Comparison of relative machine angles collected from hybrid simulator and PSS/E simulation for IEEE 39 bus test system: $\delta_{i,10} = \delta_i - \delta_{10}$; $i = 6 \ldots 9$.

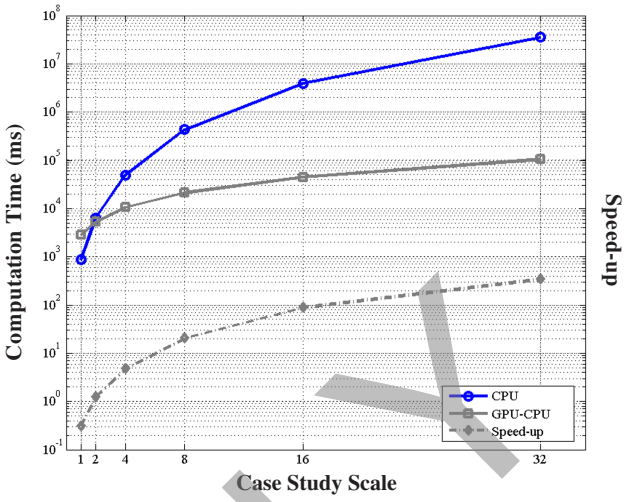

Fig. 8. Speed-up of GPU-CPU co-processing.

TABLE I
SYSTEM SIZE AND COMPUTATION TIME FOR A SIMULATION DURATION OF 1580MS

Scale	Gens	Buses	Lines	CPU	GPU-CPU	Speed-up
1	10	39	34	0.9s	2.8s	×0.3
2	20	78	71	6.4s	5.2s	×1.2
4	40	156	145	49.8s	10.5s	×4.7
8	80	312	302	7.2m	21.1s	×20.5
16	160	624	616	1hr	44.8s	×80.3
32	320	1248	1244	10hr	1m44.4s	×344.8

algorithm which is non-parallel, then the speed-up using p parallel processors is less than $1/[f + (1-f)/p]$ [10]. Since the number of SPs being used in the GPU at any given instant varies depending on the number of threads being executed, therefore, Amdahl's law does not strictly apply to SIMD-processing in the GPU. This law, however, can still be used to predict the maximum achievable speed-up. The mean value for f in our case study is 0.21%, which means a large portion of the algorithm has been parallelized. Therefore the maximum speed-up predicted by Amdahl's law, as $p \to \infty$, is 476.2, while the speed-up achieved in our simulations is 344.8.

Another useful observation found from the achieved results is the scalability of the proposed hybrid simulator. A system whose performance improves after adding specific hardware component, proportionally to the capacity added, is said to be a scalable system. In a single GPU expanding the size of data-under-process asks for co-operation of more SPs which translates to adding more computing resources. In our experiments the size of test systems and hybrid simulator's elapsed computation time change approximately at the same rate. In the CPU-only simulation cases, however, the computation time increases at a rate that is approximately the cube of the system size increment rate.

V. CONCLUSION

This paper demonstrates how an off-the-shelf inexpensive GPU can provide significant speed-up of large-scale transient stability simulations using SIMD-based parallel computing. The accuracy of the hybrid GPU-CPU simulator was verified by comparing simulation results with those obtained from the PSS/E software. The maximum discrepancy in the generator angles resulted in GPU-CPU and PSS/E simulations was found to be 1.46%. Although we used the standard method for transient stability solution, the achieved efficiency is substantial. In the largest system used in this study the hybrid GPU-CPU simulator is more than 340 times faster than CPU-only simulation. Another advantage of the GPU, investigated in this paper, is that its computation time increases linearly when the data size expands. This is due to the GPU's massively parallel architecture. Since the numerical methods used in this study are common to most power system studies, the application of this approach can be extended to load flow, harmonics, fault and electromagnetic transient simulations.

ACKNOWLEDGMENT

The authors gratefully acknowledge the donation of a GeForce GTX 280 graphics card by D. Luebke (NVIDIA).

APPENDIX I
SYSTEM MODEL FOR THE TRANSIENT STABILITY ANALYSIS

The detailed model of a synchronous generator used in this paper is given here.

1) Equations of motion (swing equations or rotor mechanical equations): In transient stability studies it is assumed that mechanical torque (T_m) is constant during the transient phenomena, and is the negative of the steady-state value of the electrical torque ($T_m = -T_e(0)$). Therefore, the turbine and governor systems are not modeled for the transient duration.

$$\dot{\delta}(t) = \omega_R . \Delta\omega(t), \tag{7}$$

$$\dot{\Delta\omega}(t) = \frac{1}{2H}[T_e(t) + T_m - D.\Delta\omega(t)].$$

2) Rotor electrical circuit equations: This model includes two windings on the d axis (one excitation field and one damper) and two damper windings on the q axis.

$$\dot{\psi}_{fd}(t) = e_{fd}(t) - R_{fd}i_{fd}(t), \tag{8}$$

$$\dot{\psi}_{1d}(t) = -R_{1d}i_{1d}(t),$$

$$\dot{\psi}_{1q}(t) = -R_{1q}i_{1q}(t),$$

$$\dot{\psi}_{2q}(t) = -R_{2q}i_{2q}(t).$$

3) Excitation system: Fig. 9 shows a bus-fed thyristor excitation system, classified as type $ST1A$ in the IEEE standard [11]. This system includes an AVR and PSS.

$$\dot{v}_1(t) = \frac{1}{T_R}[v_t(t) - v_1(t)], \tag{9}$$

$$\dot{v}_2(t) = K_{stab}.\Delta\omega(t) - \frac{1}{T_w}v_2(t),$$

$$\dot{v}_3(t) = \frac{1}{T_2}[T_1\dot{v}_2(t) + v_2(t) - v_3(t)].$$

4) Stator voltage equations:

$$e_d(t) = -R_a i_d(t) + L''_q i_q(t) - E''_d(t), \tag{10}$$

$$e_q(t) = -R_a i_d(t) - L''_d i_d(t) - E''_q(t),$$

where

$$E''_d \equiv L_{aq}[\frac{\psi_{q1}}{L_{q1}} + \frac{\psi_{q2}}{L_{q2}}], \tag{11}$$

$$E''_q \equiv L_{ad}[\frac{\psi_{fd}}{L_{fd}} + \frac{\psi_{d1}}{L_{d1}}].$$

5) Electrical torque:

$$T_e = -(\psi_{ad}i_q - \psi_{aq}i_d), \tag{12}$$

where

$$\psi_{ad} = L''_{ad}[-i_d + \frac{\psi_{fd}}{L_{fd}} + \frac{\psi_{d1}}{L_{d1}}], \tag{13}$$

$$\psi_{aq} = L''_{aq}[-i_q + \frac{\psi_{q1}}{L_{q1}} + \frac{\psi_{q2}}{L_{q2}}].$$

where ω_R, H, D, R_{fd}, R_{1d}, R_{1q}, R_{2q}, R_a, L_{fd}, L_{d1}, L_{q1}, L_{q2}, L''_d, L''_q, L_{ad}, L_{aq}, L''_{ad}, L''_{aq}, T_R, T_w, T_1, T_2, and K_{stab} are constant system parameters whose definition can be found in [12].

According to this formulation the vector of state variables in (1) and (2) of the synchronous generator is:

$$x = [\delta\ \Delta\omega\ \psi_{fd}\ \psi_{1d}\ \psi_{1q}\ \psi_{2q}\ v_1\ v_2\ v_3]^t. \tag{14}$$

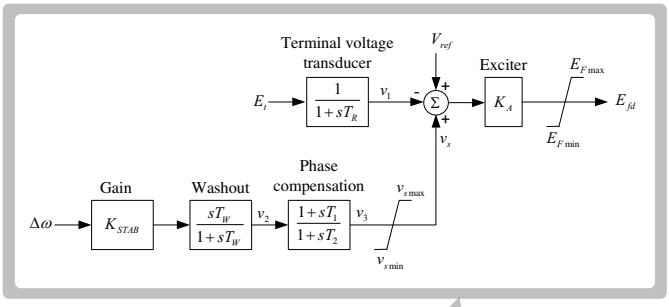

Fig. 9. Excitation system with AVR and PSS.

REFERENCES

[1] B. Stott, "Power system dynamic response calculations", *Proc. of IEEE,* vol. 67, no. 2, July 1979, pp. 219-241.

[2] J. Q. Wu, A. Bose, J. A. Huang, A. Valette, F. Lafrance, "Parallel implementation of power system transient stability analysis", *IEEE Trans. Power Syst.,* vol. 10, no. 3, August 1995, pp. 1226-1233.

[3] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, "GPU computing", *Proc. of the IEEE,* vol. 96, no. 5, May 2008, pp. 879-899.

[4] T. Wong, C. Leung, P. Heng, J. Wang, "Discrete wavelet transform on consumer-level graphics hardware", *IEEE Trans. Multimedia,* vol. 9, no. 3, April 2007, pp. 668-673.

[5] H. Schive, C. Chien, S. Wong, Y. Tsai, T. Chiueh, "Graphic-card cluster for astrophysics (GraCCA) Performance tests", *New Astronomy,* vol. 13, no. 6, August 2008, pp. 418-435.

[6] NVIDIA, "NVIDIA GeForce 200 GPU architectural overview", May 2008, pp. 1-23.

[7] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture", *IEEE Micro*, vol. 28, no. 2, March-April 2008, pp. 39-55.

[8] NVIDIA, "CUDA CUBLAS library", March 2008.

[9] P. M. Anderson, A. A. Fouad, *Power system control and stability*, Iowa State University Press, 1977.

[10] D. Blythe, "Rise of the Graphics Processor", *Proc. of the IEEE,* vol. 96, no. 5, May 2008, pp. 761-778.

[11] IEEE Std 421.5-2005, "IEEE recommended practice for excitation system models for power system stability studies", *IEEE Power Eng. Soc.,* April 2006, pp. 1-85.

[12] P. Kundur, *Power system stability and control*, McGraw-Hill, 1994.

Vahid Jalili-Marandi (S'06) received his B.Sc. and M.Sc. degrees in Power Systems Engineering from the University of Tehran, Iran, in 2003 and 2005, respectively. Currently, he is a Ph.D. candidate in University of Alberta, Canada. His research interests include transient stability studies, and digital real-time simulations in power systems.

Venkata Dinavahi (M'00) received his Ph.D. in Electrical and Computer Engineering from the University of Toronto, Canada, in 2000. Presently, he is an associate professor at the University of Alberta. His research interests include electromagnetic transient analysis, power electronics, and real-time simulation and control.