Electromagnetic Transient Simulation of Large-Scale Electrical Power Networks Using Graphics Processing Units

Jayanta Kumar Debnath, Aniruddha M. Gole, Wai-Keung Fung

***Abstract*-- This paper presents the application of graphics processing unit (GPU) based computing to speed-up electromagnetic transient (EMT) simulation of large power systems. In this scheme the GPU is mainly deployed to perform the computationally intensive part of the simulation in parallel using its built-in massively parallel processing cores, and the CPU (conventional central processing unit) is assigned for sequential jobs like, flow control of the simulation, temporary memory storage of output variables and so on. To demonstrate the methodology, the paper uses an electromagnetic transient simulation model of the IEEE 39 Bus as the basic kernel. Larger test cases are created by interconnecting several IEEE 39 Bus systems. Results show that with a hybrid environment consisting of GPUs and CPUs, simulation time is greatly reduced compared to the traditional CPU-only implementation.**

***Keywords*: Electromagnetic transient simulation, GPU-computing, parallel processing, Power systems simulation, CUDA-C programming.**

I. Introduction

Power systems are complex networks consisting of diverse components such as generators, loads, transmission lines, transformers, power-electronic switches, etc and generally occupies vast geographical area. Analytical solution of these large and complex systems is usually impossible. Therefore, numerical simulation is often the only way to obtain solutions [1], [2], [3]. Several different simulation tools, each with its own applicable frequency bandwidth, can be used for the design and operation of modern power systems [1], [4], [5], [6], [7]. Electromagnetic transient (EMT) simulation is most comprehensive tool, and models the full range of frequencies, from very slow transients in electric machines to fast transients such as those caused by circuit breaker operations, power electronic converters and lightning [2], [3], [8], [9], [10]. In early days (i.e. before the digital computer based EMT simulations), transient network analyzers (TNAs) [9], [11], [12] were used for EMT simulation. With the introduction of digital computers, many techniques have been used to solve the electromagnetic transient related problems. The digital computer based electromagnetic transient simulation program (EMTP) for multi-node networks was first introduced by H.W. Dommel [2], [3], [9], [13]. In comparison to other available modeling approaches, EMT simulation [3], [8], [9], [11], [13] models power system equipment in its greatest detail. Due to the inherent complexity and computational intensity of EMT simulations, it was originally used to simulate a small manageable subset of the larger network. However EMT simulation is being increasingly applied to study larger power networks with fast acting dynamic devices. Continuous increase in demand for interconnection of systems has lead to the creation of extremely large and interconnected power systems [3], [9], [14]. EMT simulation of such a complex network using CPU-based simulators is time consuming and simulation time can become excessively large. Many attempts are being made to reduce the simulation times, such as use of supercomputers, pc-clusters, etc [6], [15], [16], [17], [18]. Although improvements in simulation performance can be obtained, major drawbacks such as the cost of installation (especially in case of supercomputers) and inter-processor communication (especially in case of PC-clusters), are still of great concern. The approach presented in this paper deploys graphics processing units (GPUs) [19], [20], as a cost effective and high performance alternative [21], [22], [23] for EMT simulation of large scale power systems. In this approach a standard computer with a compute unified device architecture (CUDA) [19] enabled graphics card (GPU) is used to form a hybrid environment for simulation. In this simulation process, part of the EMT simulation with massive parallelism is offloaded to the GPU, which results in massive performance gain [19], [21], [22], [23], [24]. A particularly attractive feature of this approach is that GPUs are very inexpensive as they are mostly used in standard desktop computers for gaming and animation related applications. This provides the possibility of having a high power but extremely economical simulator.

J. K. Debnath is currently a Ph.D. candidate in the Department of ECE, University of Manitoba, Winnipeg, Manitoba, Canada (e-mail: jayanta072@yahoo.com).

A. M. Gole is a distinguished professor and NSERC industrial research chair in the Department of ECE, University of Manitoba, Winnipeg, Manitoba, Canada (e-mail: Aniruddha.Gole@ad.umanitoba.ca).

W. K. Fung is with the Department of ECE, University of Manitoba, Winnipeg, Manitoba, Canada (e-mail: fungwaikeung@gmail.com).

Paper submitted to the International Conference on Power Systems Transients (IPST2013) in Vancouver, Canada July 18-20, 2013.

II. Overview of GPU architecture and GPU based computing

GPUs are normally used in standard computers for display purposes with support for high performance gaming and animation related applications [24], [25], [26]. GPUs have a powerful graphics engine, which is highly parallel in

architecture and is programmable for general purpose computing [26], as shown schematically in Fig. 1. Fig. 1 shows the early (i.e. easier to understand) version of the GPU (G80) architecture. More modern GPU architectures may be found in [19]. G80 version of the GPU consists of 8-streaming multiprocessors (SMs). Each SM consists of 16-streaming processors (SP). Each SP consists of an arithmetic logic unit (ALU) capable of performing simple arithmetic operations such as addition, multiplication, division, etc as well as different logic operations. To perform more complex functions such as trigonometric, exponential or logarithmic functions there are two "super function" (SF) units in each SM. Advantage in speed can be achieved using the limited shared memory (which is equal to 16 kB in case of Geforce GTX 590 GPU) on each SM, shared between the 8 SPs. All of these streaming processors are capable of performing operations in parallel. More details on GPU architecture may be found in [19], [24], [25], [26]. The GPU accelerates computationally intensive applications by operating in a single-instruction multiple-thread (SIMT) mode [19], [24], [25]. In this SIMT mode, the same instruction is executed in parallel by multiple threads that run on identical cores [19], [24], [25].

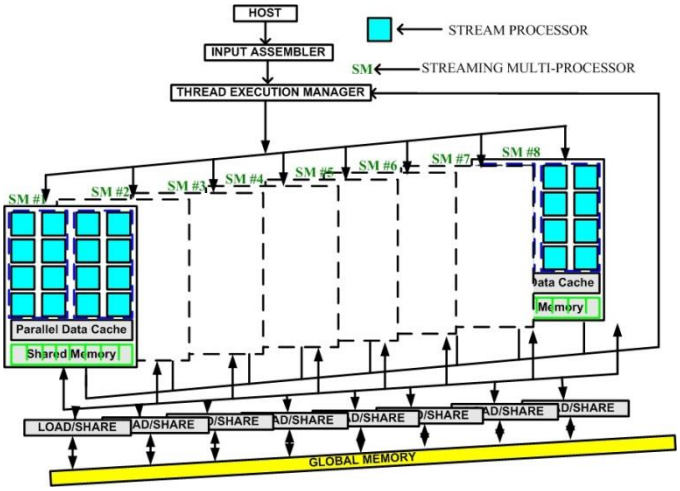

Fig. 1: Schematic of Graphics Processing Unit (GPU) architecture [24].

Fig. 2 shows the schematic interconnection of a GPU with a CPU. In this figure, both the CPU and the GPU have multiple cores. Typically a GPU has significantly larger number of cores than a CPU. For example the NVIDIA Geforce GTX-590 has 512 cores and an Intel core i7 CPU has only 4 cores. Researchers have used GPUs for general purpose computations such as real time computer vision, fluid dynamics simulations and molecular dynamics simulations [26] and electromagnetic transient simulations [21], [22].

CUDA is NVIDIA's parallel computing architecture [19], [24], [26] and CUDA based C programming is used in this work. A general CUDA-program has two parts, namely (1) a serial/sequential part and (2) parallel part. On the sequential part, no parallelism exists and the code is executed on the CPU. The second portion with massive data parallelism is implemented on the GPU.

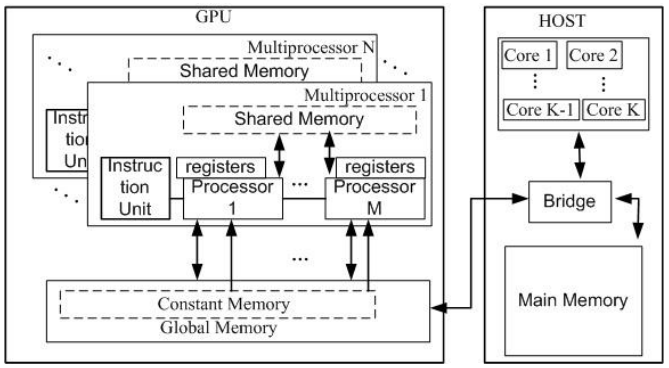

Fig. 2: Schematic of Graphics Processing Unit (GPU)-mounted PC architecture.

Fig. 3 shows the schematic execution of a typical CUDA program. The execution starts with the sequential parts executed on the CPU, and whenever a *kernel* is launched the execution is transferred to the GPU and instructions specified on this *kernel* are implemented in parallel. The CUDA-C compiler differentiates between the CPU job and the GPU job during the compilation process. The GPU portion of the program normally starts with reserved keywords, commonly known as *kernel*s. A *kernel* function generates a large number of blocks and threads to implement data parallelism. In general a *kernel* function specifies the number of blocks and threads to be generated during its launching to execute the portion of the program in parallel. Additional details on CUDA-programming models and example programs with CUDA-enabled devices may be found in [19], [24], [25], [26].

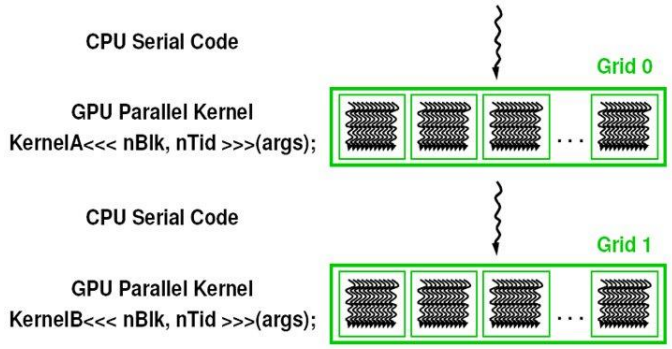

Fig. 3: Schematic of execution of a Compute Unified Device Architecture (CUDA) program [24].

III. Overview of EMT simulation

The electromagnetic transients simulator developed in this paper presently contains typical power system component, including lumped elements such as resistances (R), capacitances (C), inductances (L), coupled circuits (M) [3]; and synchronous machines. Dommel's approach [2] is used to convert the L and C elements to Norton equivalent circuit forms consisting of conductances and current sources as shown in Fig. 4. Similar equivalents may be used for the coupled circuit and distributed parameter elements, but have multiple ports instead of the single port for the Ls and Cs. These equivalent circuits allow the calculation for node voltages (typically) in a given time-step from knowledge of

the sources and 'history' terms, which represent currents/voltages from the previous time-step.

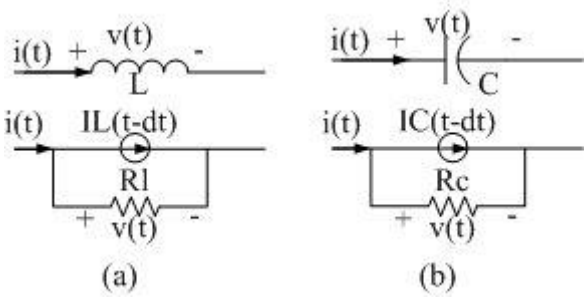

Fig. 4: Schematic equivalent circuit for (a) Inductor and (b) Capacitor following Trapezoidal Rule as proposed by Dommel [2], [3].

In this work the synchronous machine is modeled as interfaced three-phase current source following the method mentioned in [5]. Other power system equipment such as power electronic converters, etc, can also be represented in this formulation as time dependent Norton Equivalent sources [2], [3], [9], but as yet have not been included in the program. The EMT simulation process starts with time, t = 0 and with a time step of Δt. The first step is to replace all the equipment with their equivalent current source/conductance models. The resultant resistive network is solved by nodal analysis [3], [7], [13], [17], [27]. The resulting mathematical representation (using Nodal analysis [27]) has the following general form at any time *t*:

$$[Y] \times [V] = [J] - [I_H] \quad (1)$$

Where:

[Y] is the nodal admittance matrix.

[V] is the node voltage vector and [J] the vector of source currents at time t.

$[I_H]$ is the vector of history currents.

Equation (1) can be solved for the unknown nodal voltage vector $[V] = [Y]^{-1} \times [J - I_H]$. Triangular decomposition rather than explicit inversion of [Y] may be used in the solution. Once [V] is known, all currents and voltages can be computed, and the time variable *t* advanced by Δ*t*. The new values become the history values for the next time-step. The process continues until the time *t* reaches the finish time.

A. Exploiting Parallelism

EMT simulation is a highly parallel computation process [15]. One source of parallelism arises due to the finite travel time for electromagnetic signals across transmission lines. If the travel time is larger than the time-step Δ*t*, two networks connected by a transmission line can be solved independently in parallel, because in any given time-step, voltages or currents calculated in one of the networks cannot immediately affect events in the network. This approach has been studied earlier and is the basis for many of the real-time simulator algorithms [28]. However, we do not discuss this obvious parallelism in this paper, rather we see if computation gains can be made in the solution of a single subsystem using GPU computing. Any transmission lines present are included for the moment, as pi-sections in the same subsystem.

Many attempts have been taken to speed up EMT-simulation using parallel processing techniques such as mentioned in [15], [18], [29], [30]. Due to the involvement of large number of computations involving floating point variables, matrix vector multiplication consumes a large amount of the time in EMT-simulation. For example, for a system with 390 buses (duration of simulation was 10 sec) on the Intel core i7 processor, matrix vector multiplication required 644.18 seconds, where the total time for the whole simulation was only 686.11 seconds, details on this will be presented later. For larger systems simulation time becomes worse. In these simulations there were only sporadic openings of switches, as is the case for switching transient studies in ac networks. If power electronic devices are present, the switching events become large, and the corresponding re-factorization times would be large, but this case is not considered in this paper. This matrix vector multiplication is a highly parallel task [30], ideally suited for the GPU. The same system of 390 buses required only 23.39 seconds to finish the matrix vector multiplication and the total simulation time was reduced to 27.81 seconds from the previous 686.11 seconds. Similar parallelism exists in the computation of the history terms and transmission lines (pi-section) related computations. These highly parallel jobs are also positively affected by the power of GPU-computing.

IV. EMT SIMULATION USING GPU-COMPUTING

Special architecture of the GPU makes it capable of handling massively parallel computations on its onboard parallel processors [26]. Mainly, there are two commonly used programming-packages to program GPUs for general purpose computing:

- OpenCL [24] - an industry-wide standard with programming style that resembles OpenGL;
- CUDA for C/C++/Fortran, which is specific to NVIDIA GPUs [19].

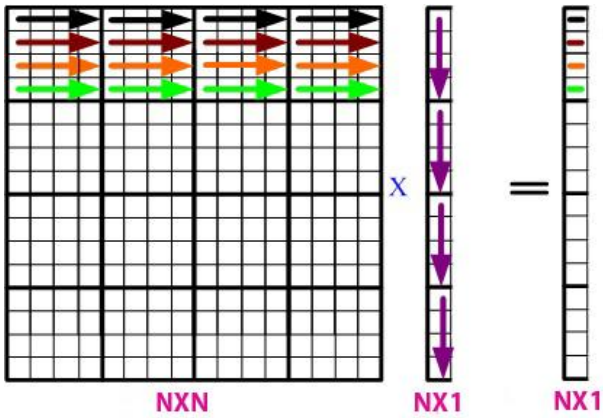

Fig. 5: Schematic of parallelism in matrix-vector multiplication using different blocks.

In this work the CUDA-C programming is used. In GPU-computing, GPU acts as a helper to the main CPU. All the control instructions originate on the CPU and are passed to the GPU, which is only responsible for executing these instructions. Due the communication overhead, there is some

lost time. Due to this inherent time-offset, using the GPU for all computations is contra-recommended where small networks are concerned, [24], [26]. This is because the communication overhead is large as compared to the computation time.

On the other hand, when the computation effort becomes large on the GPU as is the case with larger networks, performing matrix-vector multiplication on the GPU shows extensive speed up in computations [21], [22]. Reference [21], shows that if only the matrix-vector multiplication is performed on the GPU, at-least a network with 60 nodes is required for the GPU based computing to be faster than CPU based computing.

A. Matrix-Vector Multiplication on the GPU

In this case, the whole matrix and the vector is divided into different blocks (as shown in Fig. 5), taking few rows in each block. The number of threads (or rows) per block on the GPU is fixed by the CPU during the launching of the kernel (which is ultimately fixed by the programmer). Special keywords such as *threadIdx.x* and *blockIdx.x* are used to identify different blocks and threads in the computation. More details on parallel implementations of the matrix-vector multiplication may be found in [24], [25], [31].

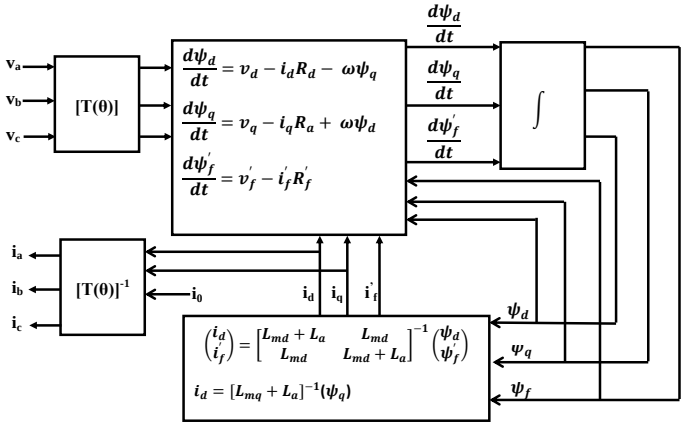

Fig. 6: Schematic of the generator model as used in this work [3, 5].

B. Simulation of Synchronous Generators on the GPU

Fig. 6 shows the schematic of the generator model (as used in this work). This generator related computations are performed in the *dq0* domain. Details on generator modeling may be found in [3], [5], [32]. Generator related computations are also highly parallel and suitable for implementation on the GPU. Typically generators require three parallel threads (for three phase system) and each generator may be assigned one block to perform the computations in parallel.

C. History current related computations on the GPU

Trapezoidal rule based numerical integration technique for power systems [2], [3], introduces history current terms for inductive and capacitive branches (as shown in Fig. 4). These history current computations are also highly parallel. To implement these computations on the GPU three threads are assigned for a three-phase system for each inductive or capacitive element. Several of these inductive or capacitive branches are combined in one block to perform the computations in parallel.

D. Current vectors updating on the GPU

EMT simulation requires updating the nodal injection current vector in each time step. This part of the admittance matrix based EMT-simulation has the least amount of parallelism. For an *N* phase power system *N* different threads could be launched to perform these computations in parallel on the GPU. It has been shown in [22] that performance is higher in case all the computations performed on the GPU. Even though the degree of parallelism is small, keeping the calculation on the GPU avoids communication overhead in passing information to the CPU. Hence, in this work all current vectors related computations are performed on the GPU.

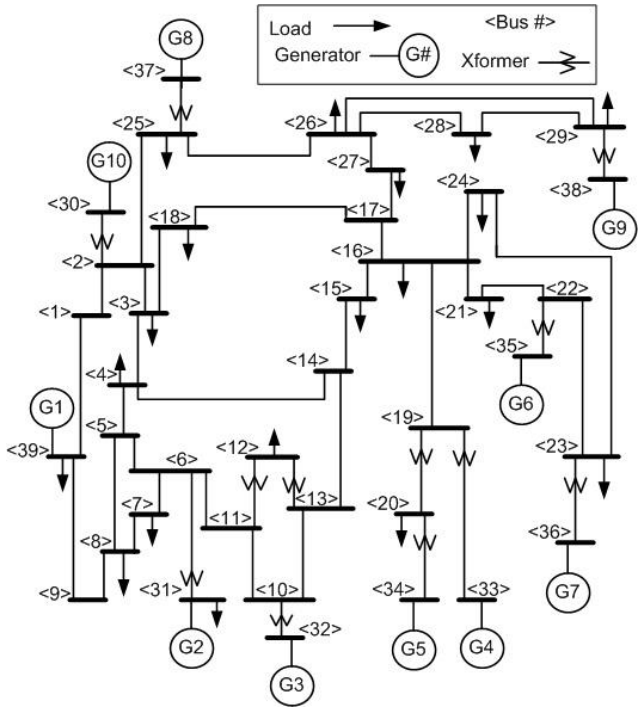

Fig. 7: Schematic of IEEE 39 Bus system.

V. SIMULATION RESULTS

Details of the hybrid platform (used in this work) are listed in Table I. The computer used in this simulation has an Intel Core i7 2600K processor (3.40GHz) with total RAM of 16GB. The GPU is NVIDIA GTX GeForce 590. The GPU is connected to the CPU through a built-in PCIe bus on the motherboard. The operating system of the machine is Linux (distribution Fedora 14) [33]. In the results reported here, to simplify the simulation scenarios, switching events were not simulated. The system was turned on with all sources connected and allowed to transition into its steady state as the transients decayed. Fig. 7 shows the schematic single line diagram of an IEEE 39 Bus system. This system is taken as the basic building block to implement larger test systems for simulation (i.e. Fig. 7).

TABLE I
DETAILS OF THE HYBRID SIMULATION PLATFORM

Main Computer (CPU) details	
Type	Intel core i7 CPU 2600K
CPU speed	3.40 GHz
Total RAM	16GB
GPU Details	
Type	NVIDIA GeForce GTX 590
Number of multiprocessors	16
Number of cores	512
Global memory	1.5GB
Constant memory	65KB
Shared memory per block	64KB
Registers available per block	32768
Warp size	32
Max. No. of threads per block	1024

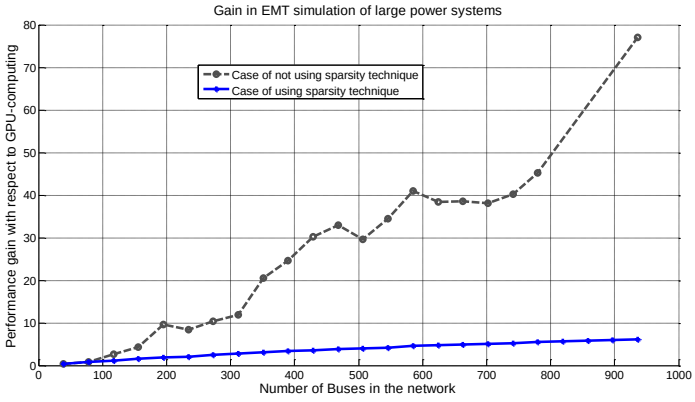

Fig. 8: Performance of GPU-computing for different power systems.

In this work two different programs for each network are developed. The first is written in conventional ANSI C code and runs sequentially on the CPU. The second program is written in CUDA-C to run in parallel on the several processors of the GPU. The duration of simulation in all the test cases was 10 seconds. Simulation results for networks of different sizes are listed in Table II. The first column in Table II indicates the 'number of buses' in the network, the second column shows the 'CPU time', i.e. the time taken for conventional sequential processing using only the CPU; and the third column shows the 'GPU time', which is GPU-computing based simulation time. For example, in case of 936-bus system, conventional ANSI C based sequential implementation requires 4530.38281 seconds (more than an hour), where as GPU-based implementation requires only 58.83000 seconds (less than a minute).

The speed up factor for GPU-computing, ($\beta_{GPU-CPU}$) is defined by the following equation [19], [21], [23]:

$$\beta_{GPU-CPU} = \frac{CPU\ only\ processing\ time}{GPU\ only\ processing\ time} \quad (3)$$

Fig. 8 shows the speed up of GPU-computing for the simulation results shown in Table II. In this case the maximum speed up is approximately 80 for a system with 936-buses. The test systems of Fig. 8 have electrically modeled generators, transmission lines were modeled using pi-networks and the transformer's magnetizing effects were not included in this simulation. It is to be noted that in this simulation process (i.e. the results presented in Table II) network sparsity was not taken advantage of. As seen from Table II, a significant speed up factor of about 80 was recorded using GPU-computing for a system with 936 buses. The sequential processor using only the CPU required more than an hour whereas the CPU-GPU based hybrid implementation required only one minute to finish the job.

TABLE II
SIMULATION RESULTS FOR DIFFERENT NETWORKS FOR SIMULATION DURATION OF 10 SEC

Number of Buses	CPU Time	GPU Time
	Total Simulation Time in Sec	
39	8.070000	19.090000
78	22.420000	26.709999
117	42.689999	16.209999
156	78.029999	17.799999
195	188.039993	19.590000
273	236.550003	22.760000
312	288.179993	24.120001
351	543.229980	26.420000
390	686.109985	27.809999
429	945.220886	31.280001
468	1102.944214	33.450001
546	1293.304688	37.570000
585	1602.443115	39.080002
780	3049.349609	67.410004
897	4141.088867	67.010002
936	4530.382812	58.830002

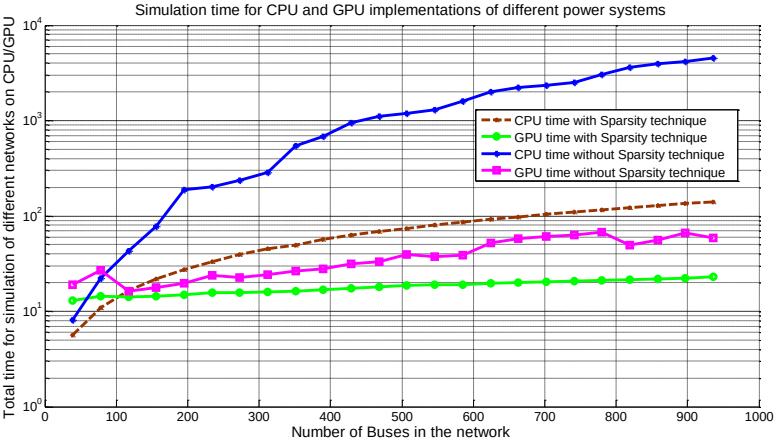

Fig. 9: Execution times for different power system sizes.

Next to investigate the speed up obtained with sparse matrix-techniques; two different sets of program (one for the CPU and the other for the GPU-implementation) were developed for each test case. In this case, a look-up table type technique (i.e. a table containing the indices for nonzero elements in the matrix was generated) is used to avoid the unnecessary multiplications involving zeros in the matrix-

vector multiplication process. Simulation results for this approach are shown in Table III. The duration of simulation was 10 sec for each case as before. For example, for the 936-bus system, conventional ANSI C based sequential implementation considering sparsity in the matrix-vector multiplication, required 140.821884 seconds, where as GPU-based implementation considering (sparsity included) required only 22.830000 seconds. Note that with sparsity, the CPU only solution itself became much faster. However, the GPU based simulation was still significantly faster than the CPU based simulation for large system sizes, but peaked at a speed-up factor of 7 for the 936 bus system. The speed up factor with sparsity is also shown in Fig. 8. Execution times listed in Table II and Table III are plotted in Fig. 9.

TABLE III
SIMULATION RESULTS FOR DIFFERENT NETWORKS WITH SPARSITY CONSIDERATION

No. of Buses	CPU Time	GPU Time
	Total simulation time in sec	
39	5.680000	13.090000
78	10.950000	14.310000
156	21.799999	14.340000
195	27.410000	14.820000
273	39.230000	15.640000
312	45.060001	16.000000
351	49.660000	16.129999
429	62.645493	17.520000
507	73.697334	18.650000
585	86.360039	18.910000
624	91.971237	19.549999
663	97.462410	19.920000
702	104.123833	20.350000
741	109.835045	20.750000
780	115.996368	21.010000
819	121.767609	21.410000
858	128.628601	21.889999
936	140.821884	22.830000

VI. Conclusions

A novel approach to perform and speed up EMT-simulation using GPUs is presented in this paper.

It is seen that speed up increases with the network size. So far systems with detailed model of generators have shown a speed up of 80 for a network of 900-buses in the network (approx.).

A typical sparse-matrix technique is introduced in this work, to avoid the multiplication involving zeros of the admittance matrix. Inclusion of this sparse-matrix technique also shown a significant speed up compared to the conventional sequential algorithm running on the CPU. The speedup with GPU was still significant; however, it was only 7 times faster for the 936 bus system.

General desktop computers equipped with these graphics cards (which is normal in these days due to the increased demand for graphics and animation related applications) could be used to perform EMT-simulation in a much more economic way.

In the near future, detailed model for different power system equipments such as transmission lines, transformers, power electronics devices, etc will be included in the simulation process. It is expected that exploration of parallelism in those power system equipments will result in more speed up in the simulation process.

In future, standard electrical networks with more buses and detailed model for different equipments will be included in the simulation process.

Also, the effect of using multiple GPUs in sharing the computation of a very large system will be investigated in future.

VII. References

[1] A. Gole, "Simulation tools for system transients: an introduction," IEEE Power Engineering Society Summer Meeting, Seattle, WA, USA, vol. 2, pp. 761–762, 2000.

[2] H. W. Dommel, "Digital Computer Solution of Electromagnetic Transients in Single- and Multiphase Networks," IEEE Transaction on Power Apparatus and Systems, vol. 44, no. 4, pp. 388–399, Apr. 1969.

[3] N. Watson and J. Arrillaga, Power Systems Electromagnetic Transients Simulation. London, United Kingdom: The Institution of Engineering and Technology (IET), 2003.

[4] A. Gole, "Electromagnetic Transient Simulation of Power Electronic Equipment in Power Systems: Challenges and Solutions," IEEE Power Engineering Society General Meeting, Montreal, Quebec, Canada., pp. 1–6, 2006.

[5] A.M. Gole, R.W. Menzies, H.M. Turanli and D.A. Woodford, "Improved interfacing of electrical machine models to electromagnetic transients programs," IEEE Transactions on Power Apparatus and Systems, vol. PAS-103, no. 9, pp. 2446-2451, Sep 1984.

[6] A. M. Gole, S. Filizadesh, R. W. Menzies, and P. L. Wilson, "Optimization-Enabled Electromagnetic Transient Simulation," IEEE Transactions on Power Delivery, vol. 20, no. 1, pp. 512–518, Jan. 2005.

[7] PSCAD/EMTDC, Manitoba HVDC research center, available online at (Jan 5, 2013): https://pscad.com.

[8] A. Gole, T. Sidhu, O. Nayak, and M. Sachdev, "A Graphical Electromagnetic Simulation Laboratory for Power System Engineering Programs," IEEE Transaction on Power Systems, vol. 11, no. 2, pp. 599–606, May 1996.

[9] Lou van der Sluis, Transients in Power Systems. London, United Kingdom: John Wiley & sons Ltd, 2001.

[10] W. Long, D. Cotcher, D. Ruiu, P. Adam, S. Lee, and R. Adapa, "EMTP A Powerful Tool for Analyzing Power System Transients," IEEE Computer Applications in Power, vol. 3, no. 3, pp. 36–41, Jul. 1990.

[11] J. R. Marti and L. R. Linares, "Real-Time EMTP-Based Transient Simulation," IEEE Transaction on Power Systems, vol. 9, no. 3, pp. 1309–1317, Aug. 1994.

[12] R. Kuffel, J. Giesbrecht, T. Maguire, R. Wierckx, and P. McLaren, "RTDS-A Fully Digital Power System Simulator Operating in Real Time," IEEE Communications Power and Computing Conference, WESCANEX, Winnipeg, Manitoba, Canada., pp. 300–305, May 1995.

[13] H. W. Dommel and W. S. Meyer, "Computation of electromagnetic transients," Proceedings of the IEEE, vol. 62, no. 7, pp. 983–993, Jul. 1974.

[14] H. W. Dommel, "Analysis of large power systems," available online at (March 19, 2011): http://www.archive.org/details/nasa_techdoc_19750021777.

[15] IEEE Task Force on Computer and Analytical Methods, "Parallel processing in power systems computation," IEEE Transactions on Power Systems, vol. 7, no. 2, pp. 629–638, May 1992.

[16] C. Larose, S. Guerette, F. Guaya, A. Nolet, T. Yamamoto, H. Enomoto, Y. Kono, Y. Hasegawa and H. Taoka, "A fully digital real-time power system simulator based on PC-cluster," Mathematics and Computers in

Simulation - Special issue: Modelling and simulation of electrical machines, converters and systems, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, vol. 63, no. 3-5, pp. 151–159, Nov. 2003.
[17] J. Franklin H. Branin, "Computer methods of network analysis," Proceedings of the IEEE, vol. 55, no. 11, pp. 1787–1801, Nov. 1967.
[18] Fernando L. Alvarado, "Parallel Solution of Transient Problems by Trapezoidal Integration," IEEE Transactions on Power Apparatus and Systems, vol. PAS-98, no. 3, pp. 1080–1090, May 1979.
[19] NVIDIA, "GPU computing: CUDA zone," available online at (Last accessed on March 20, 2011): http://www.nvidia.com.
[20] GPGPU forum, "General-Purpose computation on Graphics Processing Units," website: http://gpgpu.org/.
[21] J. Debnath, W. K. Fung, A. M. Gole, and S. Filizadeh, "Simulation of Large-Scale Electrical Power Networks on Graphics Processing Units," IEEE EPEC, Winnipeg, MB, Canada, pp. 284–289, Oct. 2011.
[22] J. Debnath, W. K. Fung, A. M. Gole, and S. Filizadeh, "Electromagnetic Transient Simulation of Large-Scale Electrical Power Networks Using Graphics Processing Units," IEEE CCECE, Montreal, QB, Canada, pp. 1-4, May 2012.
[23] V. Jalili-Marandi and V. Dinavahi, "SIMD-Based Large-Scale Transient Stability Simulation on the Graphics Processing Unit," IEEE Transactions on Power Systems, vol. 25, no. 3, pp. 1589–1599, Aug. 2010.
[24] D. B. Kirk and W. mei W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach . 30 Corporate Dr, Suite 400, Burlington, MA 01803, USA: Elsevier Inc, 2010.
[25] J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming. Boston, MA 02116, USA: Addison-Wesley Professional, 2010.
[26] John D. Owens and Mike Houston and David Luebke and Simon Green and John E. Stone and James C. Phillips, "GPU Computing: Graphics Processing Units–powerful, programmable, and highly parallel–are increasingly targeting general-purpose computing applications," Proceedings of the IEEE, vol. 96, no. 5, pp. 879–899, May 2008.
[27] Robert L. Boylestad, Introductory Circuit Analysis. Prentice Hall, 2002.
[28] R. Kuffel, J. Giesbrecht, T. Maguire, R.P. Wierckx and P. McLaren, "RTDS-A Fully Digital Power System Simulator Operating in Real Time," IEEE Communications Power and Computing Conference, WESCANEX, Winnipeg, Manitoba, Canada, pp. 300-305, May 1995.
[29] D. M. Falca´o, E. Kaszkurewicz, and H. L. Almedida, "Application of Parallel Processing Techniques to the Simulation of Power System Electromagnetic Transients," IEEE Transactions on Power Systems, vol. 8, no. 1, pp. 90–96, Feb. 1993.
[30] M. Tomim, J. R. Marti, and L. Wang, "Parallel solution of large power system networks using the multi-area the´venin equivalents (MATE) algorithm," ELSEVIER, Electrical Power and Energy Systems, vol. 31, 2009.
[31] K. Hwang and F. A. Briggs, Computer Architecture and Parallel Processing. New York, USA: McGraw-Hill, 1984.
[32] D.A. Woodford, A.M. Gole and R.W. Menzies, "Digital simulation of DC links and AC machines," IEEE Power engineering Review, pp. 36–36, Jun. 1983.
[33] "GNU Operating System," available online at (Last accessed on March 20, 2011): http://www.gnu.org/.