AUTHOR(S):	DEBNATH, J.K., GOLE, A.M. and FUNG, W.-K.
TITLE:	Graphics processing unit based acceleration of electromagnetic transients simulation.
YEAR:	2015
Publisher citation:	DEBNATH, J.K., GOLE, A.M. and FUNG, W.-K. 2015. Graphics processing unit based acceleration of electromagnetic transients simulation. IEEE transactions on power delivery [online], 31(5), pages 2036-2044. Available from: https://dx.doi.org/10.1109/TPWRD.2015.2492983
OpenAIR citation:	DEBNATH, J.K., GOLE, A.M. and FUNG, W.-K. 2015. Graphics processing unit based acceleration of electromagnetic transients simulation. IEEE transactions on power delivery, 31(5), pages 2036-2044. Held on OpenAIR [online]. Available from: https://openair.rgu.ac.uk/

Graphics Processing Unit Based Acceleration of Electromagnetic Transients Simulation

Jayanta K. Debnath, *Student Member, IEEE,* and Aniruddha M. Gole, *Fellow, IEEE,* and Wai-Keung Fung, *Senior Member, IEEE*

***Abstract*—This paper presents a novel approach to speedup EMT simulation, using GPU-based computing. This paper extends earlier published works in the area, by exploiting additional parallelism inside EMT simulation. A $2D$-parallel matrix-vector multiplication is used that is faster than previous $1D$-methods. Also this paper implements a GPU-specific sparsity technique to further speed up the simulations, as the available CPU-based sparsity techniques are not suitable for GPUs. Additionally, as an extension to previous works, this paper demonstrates modelling of a power electronic subsystem. The efficacy of the approach is demonstrated using two different scalable test systems. A low granularity system, i.e. one with a large cluster of busses connected to others with a few transmission lines is considered, as is also a high granularity where a small cluster of busses is connected to other clusters thereby requiring more interconnecting transmission lines. Computation times for GPU-based computing are compared with the computation times for sequential implementations on the CPU. The paper shows two surprising differences of GPU simulation in comparison with CPU simulation. Firstly, the inclusion of sparsity only makes minor reductions in the GPU-based simulation time. Secondly excessive granularity, even though it appears to increase the number of parallel computable subsystems, significantly slows down the GPU-based simulation.**

***Index Terms*—EMT simulation, Power systems simulation, Power system modelling, GPU-computing, CUDA-C programming, parallel algorithms.**

I. Introduction

Electromagnetic transients (EMT) simulation is commonly used to study fast transients in power systems such as lightning induced transients, switching transients including transients from power electronic switches, and so on [1], [2], [3]. Many attempts have been made to speed up EMT simulation, such as: parallel implementation of the EMT algorithm [4], use of the multi area Thevenin equivalents (MATE) algorithm [5], multi-processor based parallel EMT simulation [6], etc. These approaches have improved the performance of EMT simulations. The approach presented in this paper, uses Graphics Processing Units (GPUs) [7], [8], as a high performance and potentially cost effective alternative [9], [10], [11], [12] for EMT simulation of large power systems. The authors of this paper first reported GPU-based EMT simulation in 2011 [9] and further developments of the work were reported in [10], [11]. Recently Zhou *et al.* [13] reported on another approach for GPU-based EMT simulation, known as Node Mapping Structure (NMS). The approach presented in this paper is to speedup EMT simulation using GPU-based computing and is a potential alternative to the NMS algorithm.

J.K. Debnath and A.M. Gole are with the University of Manitoba, Winnipeg, MB, Canada R3T 5V6, Email: Aniruddha.Gole@umanitoba.ca.

W.K. Fung is with the Robert Gordon University, Aberdeen, AB10 7GJ, United Kingdom.

Our current paper extends our previous work with the implementation of additional parallelization techniques to further accelerate the GPU-based EMT simulation. In this paper the EMT algorithm is explored and algorithmic steps for power systems with various equipment that may run in parallel are identified. In particular, this paper includes the following new aspects:

- Implementation of matrix-vector multiplication using parallel threads deployed in $2D$ directions.
- Implementation of parallel computations for transmission lines on the GPU and an optimized GPU-based algorithm for synchronous generators.
- Implementation of power electronic switching on the GPU platform and investigation of its effect on the performance gain.
- Implementation of sparsity based algorithm on special hardware, such as a GPU with primitive processing cores.
- Investigation of the effect of test-system granularity on the performance gain of the simulation, i.e., the tradeoff between performance gains for test systems with a larger subsystem size versus test systems with a smaller subsystem size but having more interconnecting transmission lines.

In the sections to follow, this paper discusses GPU-computing and the sequential EMT models of various equipment in a power system and the parallelization techniques used to implement them on the GPU platform. It then discusses the selection of appropriate test systems. Total computation times for the GPU based simulation of various test cases are presented and compared with the total clock times for sequential (CPU based) simulations. Finally, the performance gains for GPU based simulations of various types of systems are presented.

II. Overview of GPU-computing

GPUs have many primitive processing cores on board and are capable of performing general purpose computations in parallel. The GPU accelerates computationally intensive applications by operating in a Single Instruction Multiple Thread (SIMT) mode [7], [14], [15]. The SIMT is an extension of the commonly known Single Instruction Multiple Data (SIMD) mode of parallelism [14]. SIMD describes the execution of an instruction on various data, whereas SIMT introduces a dynamic way for thread level parallelization of the task on various data [14]. In SIMT mode an instruction is executed with different data in parallel, using multiple threads that run on the identical GPU cores [7], [14], [15]. NVIDIA's parallel computing architecture is called Compute Unified Device Architecture (CUDA) [7], [16], [14]. CUDA based C programming is used in this work.

III. Overview of EMT simulation

Time domain EMT simulation is carried out using the following commonly known Nodal equation [1], [17], [12], [18]:

$$[Y] \times [V] = J - I_H \tag{1}$$

where $[Y]$ is the nodal admittance matrix of the network, $[V]$ is the vector of unknown node voltages at instant t, $[J]$ is the vector of injected currents at the network nodes at instant t, and $[I_H]$ is the vector of history currents injected at the nodes.

The solution for equation 1 requires the calculation of the inverse of the admittance matrix. This matrix inversion is performed at the start of the simulation or when a circuit breaker or power electronic switch operates and changes the circuit impedances. In this paper, the Gauss-Jordan method for inversion of the admittance matrix is implemented on the CPU [19]. When power electronics subsystems are present, the inverses of the anticipated post-switching configurations are pre-calculated and inserted into the solution when required.

IV. EMT simulation using GPU-computing

As mentioned in Section I, many attempts have been made to reduce the total clock time for EMT simulation. The approach presented by Zhou *et al.* [13], used a special node mapping structure (NMS) to change the admittance matrix into a perfect block diagonal matrix. Although this approach has improved the performance of EMT simulations, there may be room for further acceleration. For example, the splitting of the admittance matrix into smaller subsystems require additional book-keeping effort and memory access calls during the simulation. Additionally, this approach introduces more virtual transmission lines into the system, which (as shown later), can result in a larger communication volume and also result in an additional computation-burden for the transmission lines themselves.

The principal differences between the approach of this paper and previous approaches are: *i)* An equitable distribution of the parallel tasks of the multiplication process was implemented, which intrinsically has a smaller communication overhead and so does not require any special node mapping structure. Additionally, parallelized matrix-vector multiplication with parallel threads deployed in $2D$ directions is implemented on the GPU-platform, which uses high performance shared memory on the GPU to reduce the memory access bottleneck. *ii)* History currents and generator related computations were performed on the GPU in parallel, in a more optimized way than the authors' earlier work [11]. *iii)* Transmission lines, which interconnect subsystems were modeled outside the admittance matrix framework and used parallelization

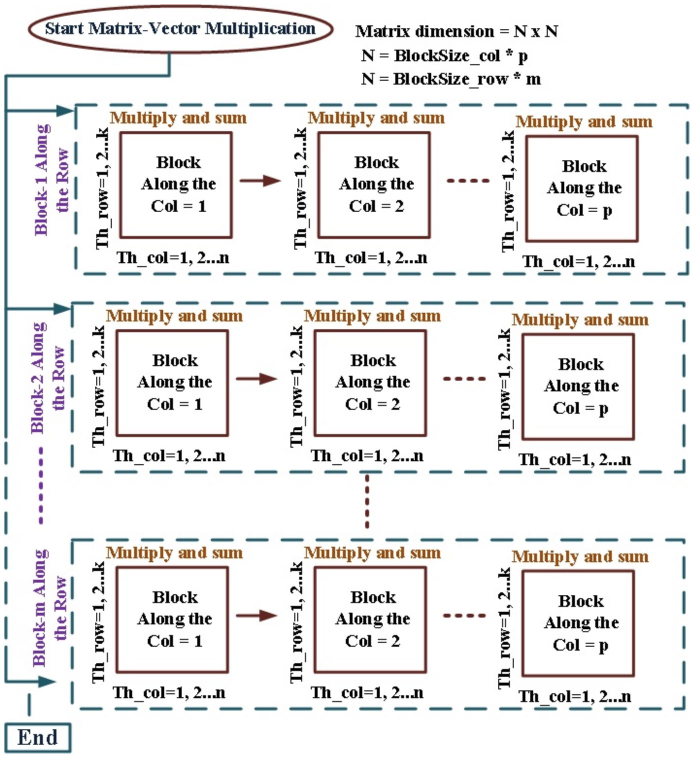

Fig. 1. Schematic of implementing GPU-based matrix-vector multiplication.

specific to GPU applications. *iv)* A sparsity based algorithm suitable for GPU-based EMT simulation was developed for the inverse-admittance matrix to further accelerate the simulation. *v)* Finally, the effect of test system granularity on the total clock times for simulation was investigated. It is shown that excessive granularity has a negative impact on the overall performance gain of the simulation using GPU-computing. It should also be noted that the parallelization algorithms used in this work are significantly different from the authors' earlier work [11] (as mentioned earlier). Details of the parallelization approaches used in this work are presented below:

A. Matrix-Vector Multiplication on the GPU with parallel threads in $2-D$ directions and using shared memory

Matrix-vector multiplication, which consumes up to 90% of the total simulation times [10], is highly parallelizable [10]. A traditional one-dimensional matrix-vector multiplication approach, where a single row is multiplied by the entire column, was implemented in the authors' earlier approaches [9], [10], [11]. By further dividing the matrix along the rows into smaller blocks allows for additional parallelism. This approach is relatively uncommon on traditional multi-core CPUs as the number of cores is much smaller, i.e., in the range of tens, than those available on a GPU, which in our case had 512 cores per GPU. In order to explain the parallelism in matrix-vector multiplication, let us consider a matrix of size $M \times N$. The total number of floating point operations required to perform the multiplication of this matrix with the vector is $M(2N-1)$. Now, if the matrix is divided into 4-parallel blocks along the rows and 5 parallel blocks along the columns (i.e. $2D$ parallelism), essentially the total number of operations will be reduced to $\frac{M}{4}(\frac{2*N}{5}-1)$ per block. As the computations for all of these 20-blocks will be performed in parallel, total clock time to perform this multiplication will be reduced accordingly. This type of $2D$ parallelism of matrix-vector multiplication is applied in this paper as opposed to one-dimensional parallelism applied in the authors' earlier works [9], [10], [11]. In order to implement parallel thread deployment in $2D$ directions, appropriate algorithmic changes have been performed in the current paper, as shown schematically in Fig. 1. This figure shows the deployment of parallel blocks along the x and y axis directions during the invocation of the *kernel*-function (i.e. invocation of a set of parallel tasks on the GPU [14], [15], [20]). In this case, parallel threads are deployed along the rows and columns simultaneously for each block.

B. History current computations on the GPU

History currents in EMT simulations [2], [1] are a function of state variables from the previous time steps. History currents from all elements such as capacitors, inductors, coupled circuits (transformers) can be simultaneously computed. In this paper parallelized history current computations are performed on the GPU. This paper optimizes history current computations by implementing a regular pattern of the necessary data-structure for history currents as compared to the authors' earlier work [9], [10], [11]. The *kernel*-function for history current computations deploys parallel threads for every block and simultaneously performs the history current computations on the GPU.

C. Simulation of Synchronous Generators on the GPU

In this work electrical generators were modelled using the method presented in [1], [21], [22]. In this case, the generators are modelled as Norton equivalent circuits having a current source in parallel with an equivalent admittance. The generator model takes the terminal voltages as inputs and calculates the currents to be injected back into the network. All the computations related to the generators are performed in the *dq0* domain. Phase domain to *dq0* domain conversions are conducted at the interface to the external EMT solver and *dq0* domain computations are used internally for the machine module. To adapt generator related computations to the GPU, all generator related data were fitted into a matrix form. Finally, this matrix was divided into several blocks containing parallel threads, which were deployed during the invocation of the *kernel*-function for generators. The generators also have a simple excitation control system as shown in Fig 2. The ac voltage magnitude (E_{line}) is measured by full wave rectification of the three phase ac voltage waveform and first order filtering. It is compared with a reference (V_{ref}) and the error (err) is passed to a PI controller. The output of the controller determines the field voltage (V_f). The control blocks are converted to state equation form and numerically integrated independently of the network solution. The measurements from the network (i.e. measured voltage) is an input to this controller model and the calculated field voltage is the output

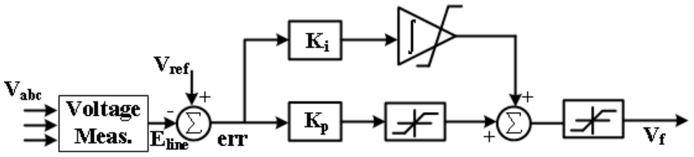

Fig. 2. Schematic of of the synchronous generator excitation control system as used in this work.

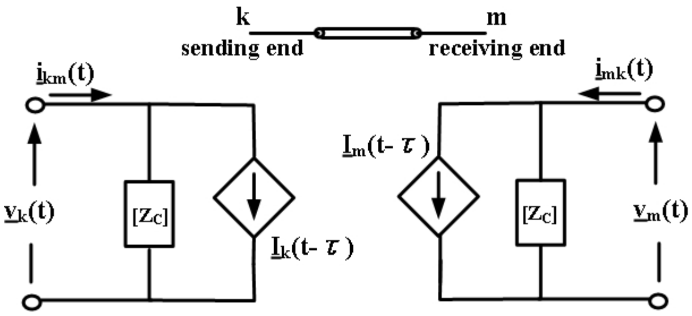

Fig. 3. Single phase transmission line model (Bergeron's model) [1], [21].

to the network solution. This approach is identical to that used in the very widely used commercial programs such as PSCAD [23] and real time simulators such as RTDS [24].

D. Current vector updating on the GPU

This part of the EMT-simulation has the least amount of parallelism. Due to low parallelism, these computations appears to be suited to being performed on the CPU in a sequential manner. However, it was shown in the authors' earlier work [10] that performing these computations on the GPU is more efficient as it does not require data transfer between the CPU and the GPU. Therefore, in this implementation, computations related to the current vector updating, were performed on the GPU with only three parallel threads.

E. GPU-based implementation of other Power System Components

1) Transmission lines: In this paper, transmission lines (TLs) have been included using a lossless Bergeron model [1], to demonstrate the methodology for GPU-based implementations. TLs are normally used to interconnect the basic test systems to create large networks. Implementation of more complex TL models on the GPU, including frequency dependent parameters will not change the basic parallelism, but will increase the computational burden of the TL model. The GPU implementation of other models for TLs are left for future work. Fig. 3 shows the schematic of a single phase TL. Multi-phase TL parameters are transformed into single-phase equivalent lines, using modal transformation matrices. They interface into the EMT solver as history terms in parallel with a Norton resistive network. The Norton network and the current sources are calculated by applying inverse modal transformation to the corresponding modal quantities [2], [17].

In this case, parallelism is achieved in two ways. Firstly, as

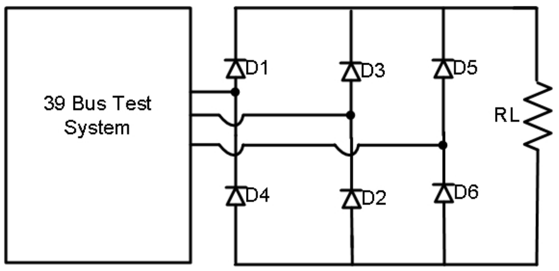

Fig. 4. IEEE 39 Bus system connected with a power electronic subsystem.

information does not propagate faster than the speed of light, if the line travel time is longer than one time-step, effectively the two sides of the TL are computationally decoupled and can be executed in parallel on the GPU. This fact is used in many real-time simulators such as [25], [26]. Also, calculations in the modal domain themselves can be performed in parallel. Both of these parallelization techniques are implemented in this work. Additionally, a special approach was taken to align the transmission line related data into a matrix form to ensure faster access to the GPU. Finally, calculations related to the receiving and sending ends of the TLs were performed in parallel using parallel *kernel*s, which ensured a further speedup in the simulation.

2) Power-Electronic Subsystems: In this work, a typical power electronic subsystem (full bridge diode based converter) was implemented to demonstrate the switching operations on the GPU-platform. A more realistic HVDC-system would require implementation of a controller model and is left for future work. Fig. 4 shows the schematic of the power electronic subsystem, where the switches (diodes) turn ON and OFF multiple times within a cycle. The switches were modeled as resistors with binary switching states (i.e. very large conductance for the ON-state and zero conductance for the OFF state) [27]. Additional logic (such as tracking the current through the switch [1]) determines the transitions between the conducting and non-conducting states of the switches (diodes). It should be noted that this basic power electronic subsystem (i.e. Fig. 4) was included in each of the building blocks presented in section V.

The converter bridge required a very minimal amount of space on the GPU memory to store the inverse admittance matrices. For example, the particular bridge of Fig. 4, results in 13 unique matrices for normal operating modes and requires about 130 kB of GPU memory (please note, each GPU of our workstation has 1.5 GB of memory on board). It should be noted that implementation of switching introduces changes in the admittances of the network, hence introduces additional admittance matrices in the simulation. Inverses corresponding to these admittance matrices were pre-computed on the CPU and stored in the GPU memory and were inserted into the simulation when the corresponding switchings occurred. This approach can be used even if a Pulse Width Modulation (PWM) type of inverter is used with multiple switchings in each cycle, because, matrices for subsystems with converters can also be pre-inverted and stored on the GPU memory. It may be possible that some configurations are missed in the pre-computation process. In that case, the CPU can be used to quickly compute the required inverses. Once computed, the new inverted admittance matrix could be properly tagged and used later in the simulation if required again. Note that this is not a real-time simulation, so occasional slowing down caused by this process is acceptable. Additionally, any subsystem with switching may be given a smaller size, in order to minimize the storage and computational burden. It is possible to model larger power electronic systems on the GPU such as bipolar HVDC transmission systems, although that would require the inclusion of phase locked loops and other control blocks that have not yet been developed by the authors.

3) Sparsity Technique implementations: There are various computationally efficient approaches (e.g., [28], [29]) to implement sparsity on a CPU, but require additional work to explore and implement the parallelism in sparsity on the GPU. Here, a commonly used lookup table [28] based approach has been applied to ignore multiplications involving zeros. Simulation based tests showed that the inverse-admittance matrix is highly sparse. Therefore, a lookup table was developed for the inverse-admittance matrix. During the simulation process an instruction reads the address of the non-zero elements from the table and the corresponding elements of the admittance matrix. Finally, multiplication with the corresponding elements of the vector is performed. Additionally, this lookup table was partitioned into various blocks involving parallel threads to perform $2D$ matrix-vector multiplications in parallel.

V. Test cases used for GPU based EMT simulation

For performance evaluation of the proposed GPU based EMT simulation, various test cases had to be scalable. Hence, rather than using distinct networks of different sizes which would all have different simulation properties, it was decided to implement a test network that could be grown in a predictable manner by adding pre-defined building blocks. It should also be noted however that the parallelization approaches presented in this paper are general and will work for any arbitrary networks. Using several different test cases, the authors' earlier work [9], showed that GPU-based EMT simulation produces exactly the same result as the commercial tools. Other researchers also used this approach for creating large test cases [5], [12], [13], [30].

Two sets of test systems with different granularity were used in this paper. Granularity is a measure of how the original network is divided into smaller clusters, where each cluster is simulated on a separate kernel. As T-lines connect various clusters together, a highly granular representation of a network will have more interconnecting lines. Hence, a quantitative measure for granularity used in this paper is the ratio of the number of TLs used to interconnect various subsystems to the total size of the network. Earlier research with real-time simulation (such as [26]) indicates that high granularity systems allow for faster simulation speeds. This paper investigates the impact of granularity on the simulation speed when GPU based simulation is used.

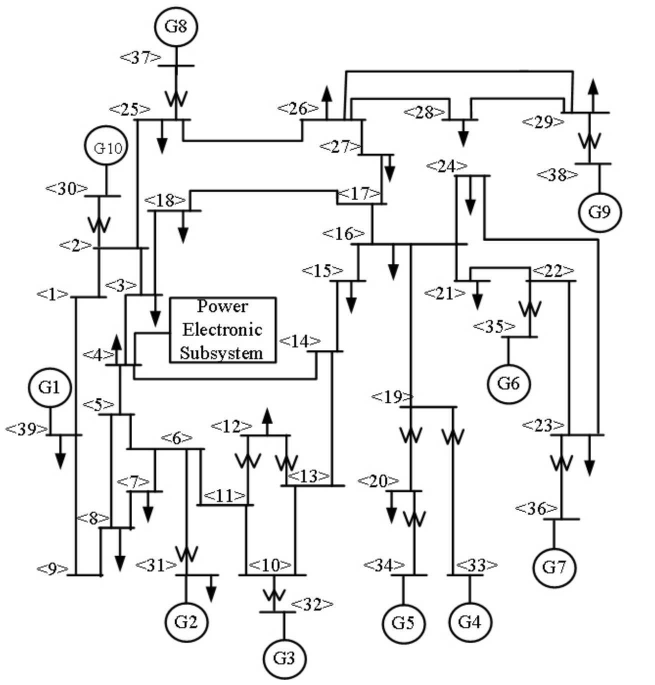

Fig. 5. Schematic of the IEEE 39 Bus system connected to the power electronic subsystem, used as *building block* 1.

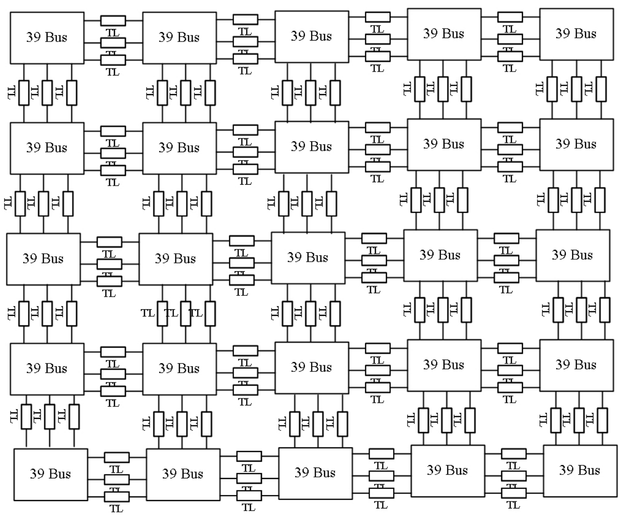

Fig. 6. Schematic of a 975 bus test system created by interconnecting building block 1 system of Fig. 5 (example of low granular test case).

A. Low Granularity Test Systems

Here, arbitrary sized test systems (with low granularity) were created by interconnecting a basic building block (shown in Fig. 5) consisting of an IEEE 39 bus system connected, to the power electronics subsystem of Fig. 4. Test systems constructed using this basic block have an asymptotic value of granularity equal to 0.154 as the network size approaches infinity (i.e. $\underset{N\to\infty}{Lim}$granularity(N) $= 0.154$, where N is the size of the network). Henceforth the system of Fig. 5 is referred to as *building block* 1. This system has 10 three phase generators, 12 three phase transformers, and one power electronic subsystem. Larger test cases were created by interconnecting the building block 1 system using TLs. As an example, Fig. 6 shows the schematic of a test system having 975 busses which was created by interconnecting 25 instances of building block 1. The system of Fig. 6 has 250 three phase generators, 25 power-electronic subsystems (i.e. 150 switches), 300 three phase transformers, and 120 three-phase T-lines.

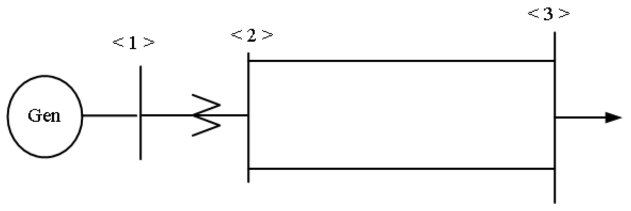

Fig. 7. The 3 bus system used in this work as building block 2.

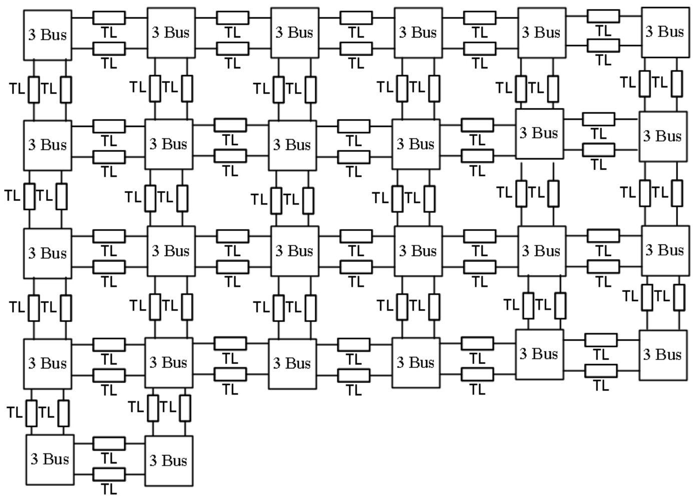

Fig. 8. Schematic of a 78 bus system created by interconnecting building block 2 system of Fig. 7 (example of high granular test system).

B. High Granularity Test Systems

A second set of test cases was created to investigate the effect of system granularity on the performance gain. A more granular system compared to the one presented above was created with a basic building block of just 3 busses with one generator and one three phase transformer as shown in Fig. 7. This system is referred to as *building block* 2. Test systems could be scaled to larger sizes by interconnecting this 3 bus system using TLs. As more and more building block 2's are added to increase the system size, the granularity approaches 1.33 in the limit. As an example, Fig. 8 shows the schematic of a 78 bus test system created using building block 2. A system consisting of 2 instances of building block 1 has 78 buses, but has only 3 interconnecting TLs. In contrast, the system in Fig. 8 also has 78 buses. However, it has 26 instances of building block 2 with a much lower computational burden compared to building block 1. On the other hand, the number of TLs interconnecting these blocks is significantly higher, namely 80. In this case, the computational burden for the interconnecting TLs (which also includes pipe-lined data

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
Shared memory per block	64KB
Warp size	32
Max. No. of threads per block	1024

transfer through the transmission line delay buffers) becomes larger than that of the system built with building block 1.

VI. PERFORMANCE EVALUATION OF GPU BASED SIMULATION

The proposed GPU-based EMT-simulation was evaluated by conducting several simulations on the various test cases presented in section V. The duration of simulation for all the test cases was 10 s and the simulation time-step used was $50\mu s$. Total clock times for the GPU based simulations were compared with the total clock times for the CPU based simulations. The algorithms to be run on the GPU and the CPU were written in CUDA-C and ANSI C respectively. The performance gain, to quantify the amount of improvement in simulation speed using the GPU based approach as compared to the CPU based approach, is quantified via the index, β_{GPU}, defined by the equation below [7]:

$$\beta_{GPU} = \frac{\text{Processing time (CPU only)}}{\text{Processing time (CPU+GPU)}}$$

The specifications of the workstation used in this work are listed in Table I. The operating system of this workstation was Linux (distribution Fedora 14) [31]. This workstation consists of an $Intel\ core\ i7\ CPU\ 2600K$ and two $NVIDIA\ GeForce\ GTX$ 590-GPUs. The GPUs are connected to the CPU through the PCIe bus on the motherboard. Each $NVIDIA\ GeForce\ GTX$ 590-GPU contains two identical GPU with 512 cores, [7]. Hence, the workstation had four GPUs with 2048-primitive cores to perform computations in parallel. However, only one GPU with 512 cores was used to implement the largest test case of this paper.

A. Typical simulation results

Simulated wave-shapes for a test case of 312-buses implemented using the proposed GPU-based EMT simulation and using the commercial tool PSCAD/EMTDC [23] are presented in Fig. 9. This 312-bus test system was created by interconnecting eight instances of building block 1 using TLs. Fig. 9 shows the DC voltage at the output of the converter, the voltages at the converter's AC bus bar, and the AC-currents entering the converter. As mentioned in section IV-E2, the diode bridge was included only to demonstrate the implementation of switching on the GPU. As can be seen from Fig. 9, GPU-computing produces essentially the same results as produced by the commercial tool, PSCAD/EMTDC. Note that the visible minor differences between the wave shapes is likely due to the slight modelling differences in transmission line models, generator models and use of interpolations in the commercial tool, PSCAD/EMTDC [23].

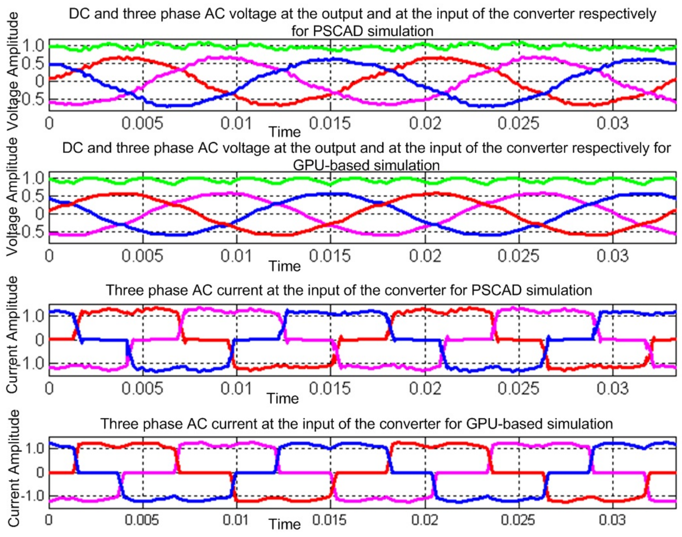

Fig. 9. Simulated DC voltage at the output of the converter, AC voltage at the input of the converter and AC currents entering the converter in a 312 bus system using commercial tool PSCAD/EMTDC and the proposed GPU-based EMT simulation.

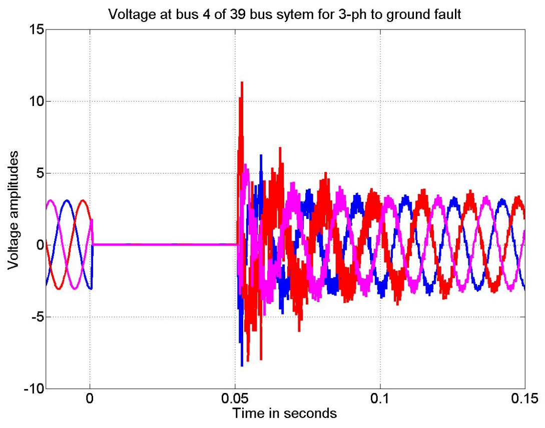

Fig. 10. Voltage at bus 24 of a 39 bus system for a 3-phase to ground fault.

Finally, Fig. 10 shows the voltage at bus 219 following a 3-phase to ground three cycle fault, in a 234 bus test system, simulated using the presented GPU-based simulation. The recovery transients are observed and balanced operation resumes after about 100 ms of fault clearance.

B. Computational performance gain for test systems with low granularity

Total clock times for simulation on the CPU in serial and on the GPU in parallel, for test cases of different sizes created by interconnecting building block 1, are shown in Table II. The number of buses and the number of T-lines for each system

TABLE II
TOTAL TIME FOR SIMULATION AND PERFORMANCE GAIN FOR TEST CASES CREATED BY INTERCONNECTING BUILDING BLOCK 1 SYSTEM OF FIG. 5, [TIME IN SECONDS]

		Without Sparsity			**With Sparsity**		
No. of Buses	No. of T-Lines	GPU Only (s)	CPU Only (s)	Gain β_{GPU}	GPU Only (s)	CPU Only (s)	Gain β_{GPU}
39	0	11.890	22.280	1.874	11.250	11.560	1.028
78	3	14.530	44.019	3.029	14.080	27.761	1.972
156	9	14.530	89.331	6.148	14.650	55.747	3.805
273	21	14.750	154.434	10.470	14.760	98.223	6.655
858	102	23.060	491.358	21.307	19.790	310.882	15.709
936	114	23.370	535.942	22.932	20.380	337.218	16.546
975	120	23.850	561.537	23.544	20.780	354.224	17.046
1365	174	28.461	784.544	27.566	24.200	494.932	20.451
1599	204	29.940	926.433	30.942	25.660	578.723	22.553
3471	474	52.407	2013.161	38.414	42.879	1261.134	29.412
3861	534	57.146	2249.380	39.362	46.888	1403.705	29.937

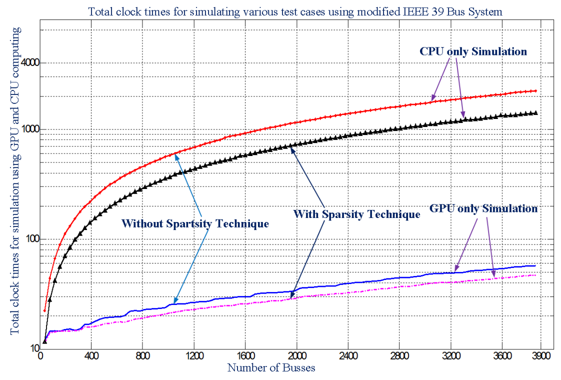

Fig. 11. Total clock times for simulating various test cases with CPU and the parallelized GPU implementations created by interconnecting building block 1 (Fig. 5).

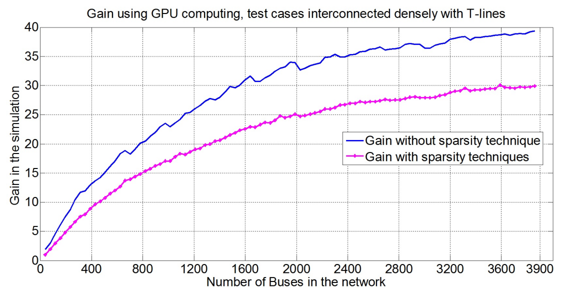

Fig. 12. Computational performance gains with GPU computing, with and without sparsity (for total clock times presented in Table II).

size are also indicated. Computation times for simulation with and without sparsity are shown in Table II. The performance gain, β_{GPU} is also shown with and without sparsity. Fig. 11 shows in a graphical form, the total clock times for the simulated cases presented in Table II. As an example of a system with 3861 busses, without sparsity, the total clock times for simulation are 57.146 s and 2249.380 s on the GPU and CPU respectively, whereas with sparsity these times become 46.888 s and 1403.705 s respectively. Therefore, Fig. 11 and Table II show that exploiting sparsity on the CPU, reduces the computation times from 2249.380 s to 1403.705 s (i.e. a

TABLE III
TOTAL TIME FOR SIMULATION FOR VARIOUS TEST CASES CREATED WITH THE MORE GRANULAR BUILDING BLOCK 2 SYSTEM OF FIG. 7, [TIME IN SECONDS]

	Test cases using 3 bus system of Fig. 7				**Test cases using IEEE 39 bus system**			
No. of Buses	No. of 3-phase T-lines	CPU times (sec)	GPU times (sec)	Gain β_{GPU}	No. of 3-phase T-lines	CPU times (sec)	GPU times (sec)	Gain β_{GPU}
39	32	16.426	15.441	1.064	0	10.430	9.739	1.071
78	80	34.011	23.484	1.448	3	23.738	13.022	1.823
117	130	51.318	32.611	1.574	6	36.290	13.232	2.743
195	226	87.761	49.335	1.779	12	60.086	13.327	4.509
234	276	103.348	58.480	1.767	15	66.642	14.406	4.626
273	324	123.364	67.356	1.832	21	87.204	14.161	6.158

speedup of 1.602). However, there is only a small benefit (i.e. from 57.146 s to 46.888 s, which is a speedup of 1.193) on the GPU. This means that the exploitation of sparsity is highly beneficial for CPU based implementations. This is as expected because the elimination of unwanted computations involving zeros greatly decreases the total number of sequential computations on the CPU. On the other hand, exploitation of sparsity on the GPU results in only a marginal benefit. Even though the implementation of sparsity on the GPU reduces the parallel multiplications involving zeros it also reduces the number of parallel threads to be deployed to perform the matrix-vector multiplication. The tasks have already been parallelized using smaller blocks which involve parallel threads, and the computational benefits from the additional sparsity are modest. Fig. 12 shows the performance gain, β_{GPU} without and with sparsity as presented in Table II. As can be seen, the speed gains for both the cases (i.e. without and with sparsity) are significantly high, but tend to a limit of about 40 without sparsity and 30 with sparsity as the system size approaches 4000 busses.

C. *Simulation based tests on high granularity test systems*

Table III presents total clock times for the simulation of various test cases created by interconnecting the building block 2 system of Fig. 7. As mentioned earlier, these systems have finer granularity compared to the systems created using building block 1. In this case the asymptotic value for granularity is 1.33, in contrast with 0.154 for the systems presented above. Table III also includes total clock times to simulate similar test cases created by interconnecting building block 1, without the power electronic subsystem (this is essentially the standard IEEE 39 bus system). Once again, the duration of the runs was 10s using a $50\mu s$ time-step. Table III lists the number of busses, the number of transmission lines and total clock times for simulating various test cases on the CPU and on the GPU. It is to be noted that the subsystem admittance matrices for the test cases created by interconnecting building block 2 are much smaller in size compared to those created by interconnecting building block 1. Therefore, it was expected that test cases with smaller subsystem sizes would run faster than the others. But the actual simulation results (Table III) are to the contrary. For example, for the more granular system with 273 busses constructed using the 3-bus building block 2, the total simulation time on the GPU was 67.356 s and on the CPU was 123.364 s. On the other hand, the total simulation time for the other system with the same number of busses constructed using the 39 bus building block 1, was 14.161 s on the GPU and 87.204 s on the CPU. This is because the increased number of TLs increases the computational burden in the test systems. It should be noted that inclusion of a frequency dependent parameter model for TLs will introduce an additional computational burden and thus give an even higher penalty for excessively granular network. Hence, proper attention must be given to the number of interconnecting TLs when partitioning big networks into smaller subsystems and the subsystems sizes. Initially, when only a few numbers of TLs are present in the network, their computational burden is much smaller than the computation time required for the admittance matrix based network solution. However, increasing the number of TLs causes the computational burden for TLs to become comparable to the admittance matrix related computations. Therefore, partitioning a large power system into arbitrarily smaller subsystems using TLs is not always efficient for GPU-based simulation.

GPU based computations can have some drawbacks. Firstly it is important to map the algorithm into groups of 32 threads (called a warp). In case the total threads in the job are not integer multiples of the warp size then there would be some warp groups having less than the maximum number of threads, which will result in poorer speed performance. This provides some programming challenges to the code designer. Also, in the rare case of extremely sequential algorithms, the CPU will outperform the GPU computations. However, in power system simulation cases, the algorithms are inherently parallelizable and so this is not a real concern. Also a great deal of work is required to transport the full set of traditional simulation tools and previously developed models to the GPUs, which ultimately require new investment, programming, developing the new platform, etc. [7], [14].

VII. Conclusions

A novel and potentially cost effective alternative to accelerate EMT-simulation using GPUs, is presented in this paper.

Simulation with several different test cases show that GPU-based computations are significantly faster, even for dense matrix-vector multiplication. Using GPU-based simulation over CPU-based simulation has yielded a speedup by a factor of 40 for system sizes approaching 4000 three phase AC buses.

The parallelization algorithm used in this paper is different from earlier GPU-based algorithms in that it inherently tried to reduce the communication burden so that special node mapping structures are not required. Other modifications such as a GPU-specific algorithm was derived and implemented, which showed a very large performance gain in the simulation speed.

A sparsity algorithm is introduced for the inverse admittance matrix to ignore computationally expensive multiplications involving zeros. A surprising result of this work is the marginal impact of implementing sparsity on the GPU-based simulations, in contrast to the CPU-based simulations where implementing sparsity has a very significant impact. This can be attributed to the fact that tasks have already been parallelized on the GPU using smaller blocks with parallel threads and the computational benefits from additional sparsity are minor.

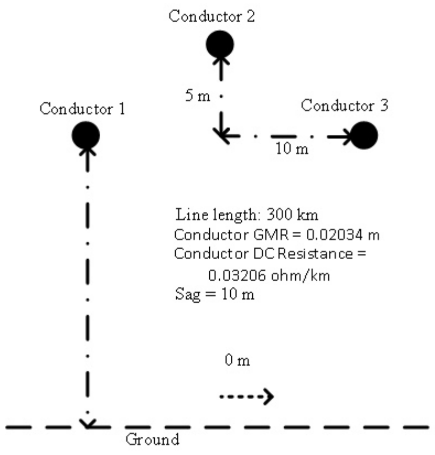

Fig. 13. Transmission line geometry as used in this work [23].

A highly efficient parallelization technique to accommodate transmission lines on the GPU was implemented. This approach also contributed to the overall performance gain in the simulation process.

The paper investigated the effect of building-block granularity on performance gains using GPU computing. Another interesting and surprising conclusion was that increasing the granularity of the test systems negatively impacts the performance gain. In these cases, the communication overhead for a large number of transmission lines cancels any advantage resulting from the smaller matrix sizes.

Appendix

Other system details are listed below:

1) IEEE 39 bus system data are as in [32].
2) All interconnecting transmission lines are three-phase non-transposed with data as in Fig. 13.
3) Resistive load for the diode bridge of *building block* 1 of Fig 4 was 825Ω.

References

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

[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] 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.

[4] 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.

[5] M. Tomim, J. R. Marti, and L. Wang, “Parallel solution of large power system networks using the multi-area thévenin equivalents (MATE) algorithm,” *ELSEVIER, Electrical Power and Energy Systems*, vol. 31, pp. 497–503, 2009.

[6] D. M. Falcáo, 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.

[7] NVIDIA, “GPU computing: CUDA zone,” available online at (Last accessed on March 20, 2013): http://www.nvidia.com.

[8] GPGPU forum, “General-Purpose computation on Graphics Processing Units,” website: http://gpgpu.org/.

[9] 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.

[10] ——, “Electromagnetic Transient Simulation of Large-Scale Electrical Power Networks Using Graphics Processing Units,” *IEEE CCECE, Montreal, QB, Canada*, May 2012.

[11] J. Debnath, A. M. Gole, and W. K. Fung, “Electromagnetic Transient Simulation of Large-Scale Electrical Power Networks Using Graphics Processing Units,” *International Conference on Power Systems Transients (IPST), Vancouver, BC, Canada*, pp. 1–4, Jul. 2013.

[12] 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.

[13] Z. Zhou and V. Dinavahi, “Parallel massive-thread electromagnetic transient simulation on gpu,” *IEEE Transaction on Power Delivery*, vol. 3, no. 29, pp. 1045–1053, Jun 2014.

[14] 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.

[15] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Boston, MA 02116, USA: Addison-Wesley Professional, 2010.

[16] 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.

[17] H. W. Dommel and W. S. Meyer, “Computation of electromagnetic transients,” *Proceedings of the IEEE*, vol. 62, no. 7, pp. 983–993, Jul. 1994.

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

[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing*. Cambridge CB2 8RU, UK: Cambridge University Press, 2007.

[20] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York, USA: McGraw-Hill, 1984.

[21] 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.

[22] 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.

[23] PSCAD/EMTDC, Manitoba HVDC research center, available online at (Last accessed on March 20, 2013): https://pscad.com.

[24] R. T. D. Simulator, available online at (Last accessed on October 2, 2014): http://www.rtds.com/index/index.html.

[25] 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.

[26] 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.

[27] T. Maguire and A. Gole, “Digital simulation of flexible topology power electronic apparatus in power systems,” *IEEE Transaction on Power Delivery*, vol. 6, no. 4, pp. 1831–1840, Oct. 1991.

[28] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*. New York: Springer-Verlag, 3rd Edition, 2002.

[29] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms)*. Philadelphia, PA 19104-2688 USA: Society for Industrial and Applied Mathematic, 2006.

[30] R. Singh, A. Gole, P. Graham, S. Filizadeh, C. Muller, and R. Jayasinghe, “Grid-processing for optimization based design of power electronic equipment using electromagnetic transient simulation,” *25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–6, May 2012.

[31] “GNU Operating System,” available online at (Last accessed on March 20, 2013): http://www.gnu.org/.

[32] T. Athay, R. Podmore, and S. Virmani, “A practical method for the direct analysis of transient stability,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-98, no. 2, pp. 573–584, Mar. 1979.

Jayanta K. Debnath received his B.Sc. (with honours) and M.Sc. in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology (BUET), in 2003 and 2006 respectively. He is currently a Ph.D. candidate in the Electrical and Computer Engineering department of University of Manitoba, Canada. He worked as a Lecturer in the Electrical and Electronic Engineering department of American International University-Bangladesh (AIUB), Dhaka, Bangladesh, during the period 2003 − 2007. His research interests include EMT-simulations, power systems analysis, network equivalent analysis, etc.

Aniruddha M. Gole (S’77-M’82-SM’04-F’10) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, India, in 1978 and the Ph.D. degree in electrical engineering from the University of Manitoba, Winnipeg, MB, Canada, in 1982, where he is currently a University Distinguished Professor.

Dr. Gole is a member of the original development team for the PSCAD/EMTDC program. He is the 2007 recipient of the IEEE PES Nari Hingorani FACTS Award. He is a Fellow of the Canadian Academy of Engineering and a Professional Engineer in the Province of Manitoba, Canada.

Wai-Keung Fung obtained his Ph.D. from the Department of Automation and Computer Aided Engineering at the Chinese University of Hong Kong in 2001. He has held post-doctoral positions at the Michigan State University and the Chinese University of Hong Hong. From 2005-2012 he was an Assist. Professor at the University of Manitoba. Currently he is a faculty member at the School of Engineering, Robert Gordon University, UK. The research interests of Dr. Fung include intelligent robotics, networked robotics, computational intelligence and human-machine interaction. He has been elevated to IEEE Senior Member since 2009.