Parallel Massive-Thread Electromagnetic Transient Simulation on GPU

Zhiyin Zhou, *Student Member, IEEE*, and Venkata Dinavahi, *Senior Member, IEEE*

***Abstract*—The electromagnetic transient (EMT) simulation of a large-scale power system consumes so much computational power that parallel programming techniques are urgently needed in this area. For example, realistic-sized power systems include thousands of buses, generators, and transmission lines. Massive-thread computing is one of the key developments that can increase the EMT computational capabilities substantially when the processing unit has enough hardware cores. Compared to the traditional CPU, the graphic-processing unit (GPU) has many more cores with distributed memory which can offer higher data throughput. This paper proposes a massive-thread EMT program (MT-EMTP) and develops massive-thread parallel modules for linear passive elements, the universal line model, and the universal machine model for offline EMT simulation. An efficient node-mapping structure is proposed to transform the original power system admittance matrix into a block-node diagonal sparse format to exploit the massive-thread parallel GPU architecture. The developed MT-EMTP program has been tested on large-scale power systems of up to 2458 three-phase buses with detailed component modeling. The simulation results and execution times are compared with mainstream commercial software, EMTP-RV, to show the improvement in performance with equivalent accuracy.**

***Index Terms*—Electromagnetic transient analysis, graphics processors, massive-thread, parallel algorithms, parallel programming, power system simulation.**

I. Introduction

THE USE of electromagnetic transient (EMT) simulation tools is no longer restricted to specialized studies that focus on analyzing the propagation of EM transients. Due to their versatility and breadth of modeling capability, offline EMT simulation tools, such as EMTP-RV [1], PSCAD/EMTDC [2], Alternative Transients Program (ATP) [3] etc., are routinely used in the planning, design, and operation of power systems, to study dynamic phenomena over a wide frequency range—from steady-state studies, such as load flow and harmonic analysis, to high-frequency studies, such as restrike overvoltages in gas-insulated substations [4]. Along with modeling and application diversity, the size of the power system simulated by EMT tools has grown concomitantly [5]. These days, it is not uncommon to simulate in detail systems containing hundreds of buses using such tools. Nevertheless, the common characteristic of the aforementioned EMT simulation tools is that they are single-thread sequential programs designed to run efficiently on single-core CPUs based on the x86 processor architecture. Throughout the 1990s and 2000s, the CPU clock speed steadily increased and memory costs decreased, fueling a sustained increase in the speed of these programs. But now with the clock speed saturated around 3 GHz due to chip power dissipation and fabrication constraints, the computer industry has transitioned to multicore CPU and many-core GPU hardware architectures, which require multithread parallel programming. Executing a single-thread EMT program on a multicore architecture is inefficient because the code is executed on a single core, one instruction after another in a homogeneous fashion, unable to exploit the full resource of the underlying hardware. The overall performance of the code can be severely degraded especially when simulating large-scale systems with high data throughput requirements. A multithread parallel code can provide substantial gain in speed and throughput over a single-thread sequential code on the multicore architecture. Even on single-core processor systems, multiple threads can add palpable performance improvement in some applications. What this means for the EMT program is that the system component models and the EMT algorithm need to be reformulated to be executed in parallel on multiple cores. Since data independence is the key requirement for efficient parallel programming, the serial component models and network data structure have to be transformed to meet this requirement. Although the focus of this paper is offline EMT simulation, it is worth mentioning that real-time EMT simulation tools are available, such as RSCAD and Hypersim, etc., which can also run offline simulations on multicore CPUs. However, these tools do require specialized expensive hardware to run real-time EMT simulation.

In this paper, we propose a GPU-based parallel massive-thread EMT simulation program (MT-EMTP). The GPU [6]–[8] has native parallel many-core processing units and high-performance floating-point number processors. To circumvent understanding the cumbersome assembly language of GPU, compute unified device architecture (CUDA) [9], a general-purpose parallel computing environment, was introduced by NVIDIA to offer an efficient C-like language to develop an application. Researchers have already begun to exploit GPUs for power system applications [10]–[18], such as data visualization, load-flow computations, electromagnetic transient, and transient stability simulations. The objective of this paper is to develop massive-thread parallel modules for

Manuscript received August 16, 2012; revised May 08, 2013, July 12, 2013, and October 22, 2013; accepted December 23, 2013. Date of publication February 17, 2014; date of current version May 20, 2014. This work is supported by the Natural Science and Engineering Research Council of Canada (NSERC). Paper no. TPWRD-00863-2012.

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2V4 Canada (e-mail: zhiyin@ualberta.ca; dinavahi@ualberta.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TPWRD.2013.2297119

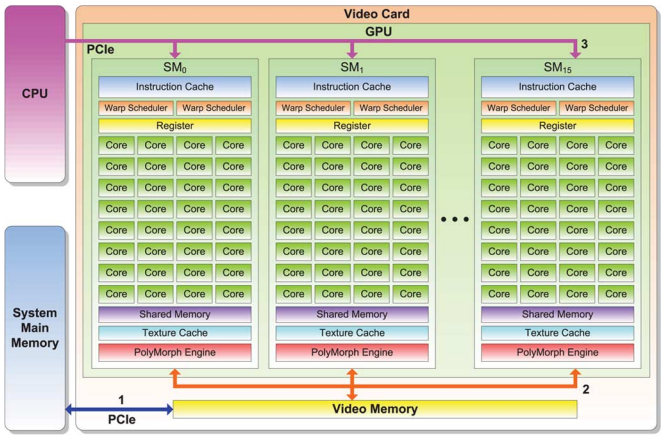


Fig. 1. GPU hardware architecture: 1) data transmission between host and device, 2) data dispatched to/from many cores, and 3) control instruction dispatch.

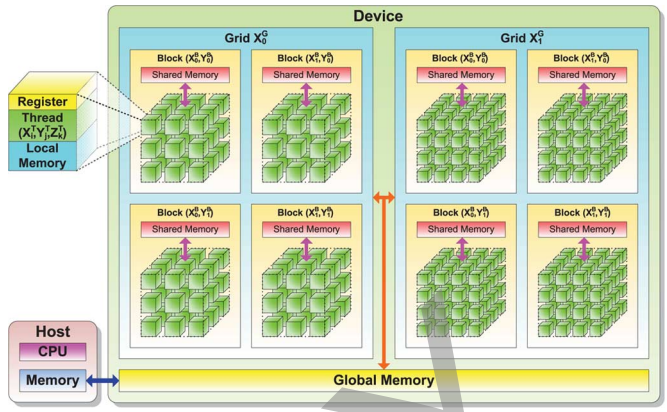


Fig. 2. CUDA abstraction.

three ubiquitous and computationally demanding system components for EMT simulation: linear passive elements (LPE), the universal line model (ULM), and the universal machine model (UMM). A new efficient node mapping structure (NMS) is proposed to reorder the original power system bus numbers using the block-node adjustment (BNA) method to obtain a block diagonal pattern for the system admittance matrix that is ideally suited for the GPU-based massive-thread parallel architecture. The performance of the developed parallel EMT program was evaluated for accuracy, computational efficiency, and scalability by using several large-scale test power systems, and compared with the EMTP-RV software program.

This paper is organized as follows: Section II describes the key features of the GPU and CUDA, which enable the design of the parallel component models and the proposed data structure used in the EMT simulation. Section III gives the details of the parallel massive-thread component models. Section IV explains the NMS using the BNA method and the sparse linear network solution. Then, the experimental results for various large-scale test systems are shown and compared with EMTP-RV. Finally, Section V gives the conclusion.

II. GPU Architecture and CUDA Abstraction

Since the most advanced concept, the Fermi architecture (shown in Fig. 1) [6] of the GPU from NVIDIA, is chosen for developing the parallel massive-thread modules for EMT simulation, it consists of several streaming multiprocessors (SMs). Each SM is.populated with many compute cores which share the registers, caches, and dedicated memory inside the SM. Since the GPU is designed to work as a coprocessor, all data and instructions come from the CPU via the PCIe interface.

In order to make every core in the GPU work efficiently, enough data must be fed to catch up to the instruction cycles; without data input, the cores can only remain idle, thus reducing computing speed. Since the GPU has far more (hundreds) cores than the CPU, the data bandwidth requirement is increased tremendously. However, the system main memory and the video memory cannot offer that ideal bandwidth; therefore, specific memory access routes have to be followed to reach the optimal speed. The data-transmission paths between the memories, that is, path 1 and path 2 in Fig. 1 are the main bottlenecks of the architecture. As explained later, the proposed massive-thread parallel EMT modules are designed to minimize the use of these paths in order to maximize computational efficiency.

CUDA is the programming tool used to implement the parallel program without dealing with the GPU assembly language. The details of the GPU hardware are taken care of by CUDA's mapping and abstraction, as shown in Fig. 2. A CUDA program (kernel) separates the hardware resources into two parts: CPU side as the *host*, on which the serial parts of the program run, and the GPU side as the *device*, on which the parallel parts of the program run.

In the CUDA thread hierarchy, Grid, Block, and Thread map to GPU, SM, and Core, respectively. However, the user need not be concerned with the actual number of GPUs, SMs, and Cores since the number of abstracted threads are automatically assigned to the physical cores in parallel or serial fashion. Therefore, even if the GPU used has fewer cores than what the CUDA program requires, the programmer can still claim the number of threads needed. The developed parallel EMT program can adapt to various CUDA-supported GPUs with a varying number of cores without change. A group of threads makes up a block, and a group of blocks makes up a grid. All threads inside a grid execute the same instruction simultaneously with multiple data input, which requires complete data independence and unified processing flow in the kernel. This is known as the single instruction multiple data (SIMD) format. On the other hand, the single-instruction multiple thread (SIMT) enables the program with thread-level parallel code for independent, scalar threads [9], thereby allowing the GPU to handle multiple branches and operations in a single instruction. The developed parallel EMT component modules and the sparse linear solver utilize the SIMD and SIMT concepts.

In the CUDA memory hierarchy, each thread has its own register and local memory. Threads of the device cannot access the host memory directly due to it being on the CPU side. Thus, all data that are processed by the GPU have to be first copied into the global memory of the device. Since the global memory (video memory) is not on-chip though it is onboard, accessing it is relatively inefficient. Shared memory, which is much faster than global memory, is offered for each block, which can be

accessed by all threads in the block. Thus, using this limited resource (48 kB/block) wisely can effectively optimize the performance of the program. The advanced Fermi architecture (compute capability 2.0 and above) offers a data cache which requires a well-organized data input to maximize access speed. In the global and shared memories, a memory address normally can only be accessed once in an instruction cycle, especially for write operation. Therefore, simultaneous multiple and random memory access should be avoided in the CUDA kernel because not all threads can guarantee that their memory access is safe. Furthermore, unlike the previous GPU generations, the Fermi architecture enables the execution of multiple kernels simultaneously for increased efficiency.

Thus, these are the aforementioned considerations of the GPU architecture and CUDA that the design of the data structure and parallel modules for massive-thread EMT simulation is based on.

III. Parallel Massive-Thread Component Models

A. Linear Passive Elements

1) Model Formulation: Linear passive elements (LPEs), such as resistance, inductance, capacitance, switches, and their combinations, are represented by a discrete-time lumped model [19]. As mentioned in Section II, since all threads in a kernel run the same instruction concurrently, a unified model is required for all LPEs in the system. Using the trapezoidal rule of integration, any LPE combination can be modeled as a discrete Norton equivalent circuit, comprised of an equivalent conductance and a history current source. In the unified model, every LPE has an R, L, or C character. An arbitrary LPE Z shown in Fig. 3(a) can be represented as an R, L, and C Thévenin equivalent circuit as shown in Fig. 3(b), where $R_{\rm eq}^{R}$ is the equivalent resistance of R, R_{eq}^{C} is the equivalent resistance of C, R_{eq}^{L} is the equivalent resistance of L, V_h^C is the capacitive history voltage, and V_h^L is the inductive history voltage. Depending on the LPE character, constant flag coefficients can be defined, for example, p^L and p^C for the L or C character. Source transformation results in the unified Norton equivalent circuit are shown in Fig. 3(c), where $G_{\rm eq}$ is the total equivalent conductance and I_h is the history current source of the unified model [21]. The LPE current $i(t)$ is given as

$$i(t) = G_{\rm eq} v(t) + I_h(t - \Delta t). \tag{1}$$

With the unified LPE lumped model, all linear elements can be processed in the same kernel.

2) Massive-Thread Parallel Implementation: Fig. 4 shows the designed parallel module for unified LPE computation. For each LPE, a CUDA thread is assigned to execute the computation based on the SIMT format. When the number n of LPEs exceeds the limitations of thread per block k, they will be divided into m groups assigned to multiple blocks

$$m = \left\lceil \frac{n-1}{k} \right\rceil + 1, \tag{2}$$

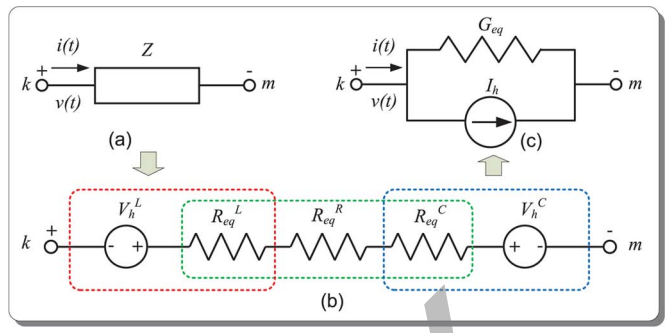


Fig. 3. Unified linear passive element lumped model.

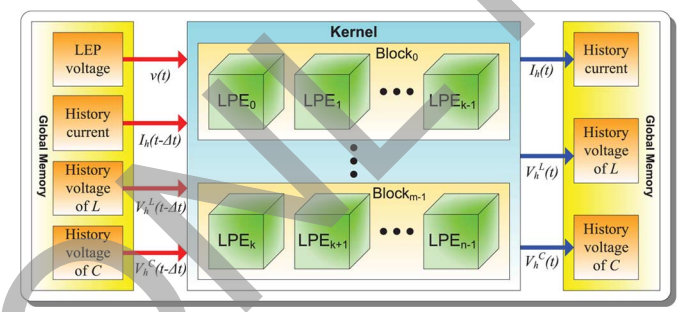


Fig. 4. Massive-thread parallel unified LPE module.

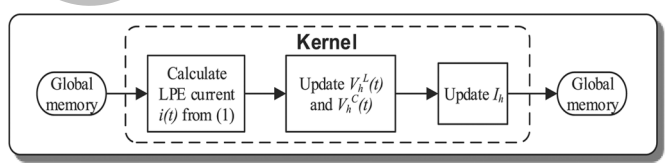


Fig. 5. Kernel operation flow in the unified LPE module.

where n, k and m are integers. The operation flow of the LPE kernel is shown in Fig. 5.

Algorithm III.1: ULPE_Kernel (k, m, n)

Assign m blocks

Assign k threads per block

Config Shared Memory

Shared Memory ← Global Memory

Task1
for each $Thread \in Block$
do $i \leftarrow G_{eq} \times v + I_h$
synchronization

Task2
for each $Thread \in Block$
do $V_h^L \leftarrow p_1^L \times V_h^L + p_2^L \times R_{eq}^L \times i$; $V_h^C \leftarrow p_1^C \times V_h^C + p_2^C \times R_{eq}^C \times i$
synchronization

Task3
for each $Thread \in Block$
do $I_h \leftarrow G_{eq} \times (V_h^L + V_h^C)$
synchronization

Global Memory ← Shared Memory

Algorithm III.1 shows the pseudocode for the CUDA kernel of the unified LPE module. Inside the kernel, there are three sequential tasks with the same CUDA thread and memory configuration. First, the LPE current $i(t)$ is computed from (1), then the inductive and capacitive history voltages $V_h^L(t)$ and $V_h^C(t)$ are calculated and, finally, the history current I_h is updated.

The only global memory accesses are reading the input variables and writing the output variables, and all computations of LPE take place inside the threads. During the EMT simulation, all variables are stored and reused on the device side; thus, the host-device and device-host data transmission is minimized in each timestep.

B. Transmission Lines

1) Model Formulation: The universal line model (ULM) is a phase-domain wideband fully frequency-dependent line model [24] capable of representing symmetrical and asymmetrical overhead transmission lines and underground cables. Traditional frequency-dependent transmission-line models [22] were constituted in the modal-domain based on real and constant transformation matrices with a frequency-dependent model for the traveling waves. These models are mainly suited for symmetrical (transposed) lines and cables. The transformation matrix of untransposed lines is, in general, complex and frequency dependent; nevertheless, such models can also be applied to untransposed conditions after appropriate numerical modifications, including eigenvector rotations, to make the coefficients in the transformation matrices real and constant. The ULM avoids the transformation matrices and is constituted directly in the phase domain; however, it involves computationally expensive convolutions.

The ULM represents the sending-end "k" and the receiving-end "m" of a line of arbitrary length, shown in Fig. 6(a), as two decoupled Norton equivalent circuits, as shown in Fig. 6(b). The ULM current $\boldsymbol{i}(t)$ is given as

$$\boldsymbol{i}(t) = \boldsymbol{G}_Y \boldsymbol{v}(t) - \boldsymbol{I}_h \tag{3}$$

where $\boldsymbol{G}_Y$ is the equivalent conductance matrix and the history currents $\boldsymbol{I}_h$ are expressed as

$$\boldsymbol{I}_h = \boldsymbol{Y} * \boldsymbol{v}(t) - 2\boldsymbol{H} * \boldsymbol{i}_r(t-\tau) \tag{4}$$

where the reflected current $\boldsymbol{i}_r$ is defined as

$$\boldsymbol{i}_r = \boldsymbol{i}(t) - \boldsymbol{i}_i \tag{5}$$

where $\boldsymbol{i}_i$ is the incident current [24].

In (4), the characteristic admittance matrix $\boldsymbol{Y}$ and propagation matrix $\boldsymbol{H}$ are approximated by the finite-order rational functions using the vector-fitting (VF) method [26]; the "$*$" denotes the numerical complex matrix-vector convolution. The numerical convolution $\boldsymbol{Y} * \boldsymbol{v}(t)$ is defined as

$$\boldsymbol{Y} * \boldsymbol{v}(t) = \boldsymbol{c}_Y \boldsymbol{x}_Y(t) \tag{6}$$

where the coefficient matrix $\boldsymbol{c}_Y$ can be obtained from the residues $\boldsymbol{r}_Y$ and the coefficients $\boldsymbol{\alpha}_Y$ which are, in turn, functions of the poles $\boldsymbol{p}_Y$; and the state variables $\boldsymbol{x}_Y$ are obtained from the coefficients $\boldsymbol{\alpha}_Y$ and line voltages $\boldsymbol{v}$. Similarly, the numerical convolution $\boldsymbol{H} * \boldsymbol{i}_r(t-\tau)$ is defined as

$$\boldsymbol{H} * \boldsymbol{i}_r(t-\tau) = \boldsymbol{c}_H \boldsymbol{x}_H(t) + \boldsymbol{G}_H \boldsymbol{i}_r(t-\tau), \tag{7}$$

where $\boldsymbol{G}_H$ is the propagation matrix of $\boldsymbol{H}$. Since the wave traveling time τ is not an integral multiple of the time step Δt normally, linear interpolation is used to approximate the reflected current $\boldsymbol{i}_r(t-\tau)$.

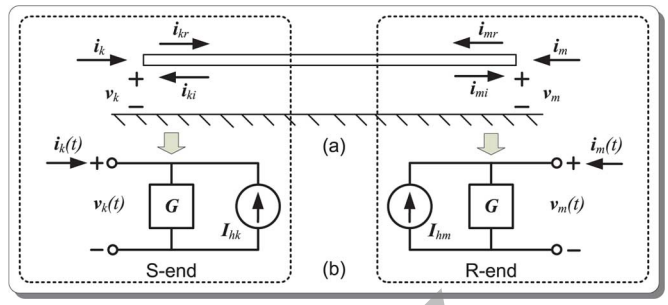


Fig. 6. Universal line model.

2) Massive-Thread Parallel Implementation: As shown in Fig. 7, the designed parallel module for the ULM includes eight kernels grouped into four stages. Stage 1 updates the reflected current $\boldsymbol{i}_r$ and calculates the interpolation for the reflected current before delay τ, Stage 2 updates the state variable $\boldsymbol{x}(t)$, Stage 3 computes the convolutions, and Stage 4 updates the incident current $\boldsymbol{i}_i$ and the history current $\boldsymbol{I}_h$. All of the kernels inside the same stage are executed concurrently in the Fermi architecture space. The computation for each ULM unit is done by a CUDA block running in SIMT, inside which multiple threads are assigned to handle vector and matrix operations based on SIMD. Therefore, every kernel has n blocks (the number of ULM units) in every stage, and the number of threads in a block depends on the dimension of computed vectors and matrices, which is typically based on the number of poles and residues from vector fitting. The data are transferred deliberately from the global memory into the shared memory first to improve the memory access performance due to the critical bandwidth requirement of vector and matrix operations.

The kernel operation flow of the ULM module is shown in Fig. 8. Since the thread dimension and shared memory size have to be reconfigured in different tasks, such as in updating variables, interpolation, and convolutions, they are separated into different kernels, and their results are output to global memory and shared with other kernels. Since the parallel computation is based on each ULM unit instead of its sending and receiving ends, all "k" and "m" variables are computed within one kernel, avoiding the data exchange between "k" and "m" ends. Similar to the LPE module, all of the module variables are limited to the device side, that is, to the global and shared memories of the GPU; thus, there is no data exchange between the host and device during ULM execution.

C. Electrical Machines

1) Model Formulation: There are several types of rotating machine models that can be used for EMT studies. The advantage of the unified machine model (UMM) [23], [25] is that

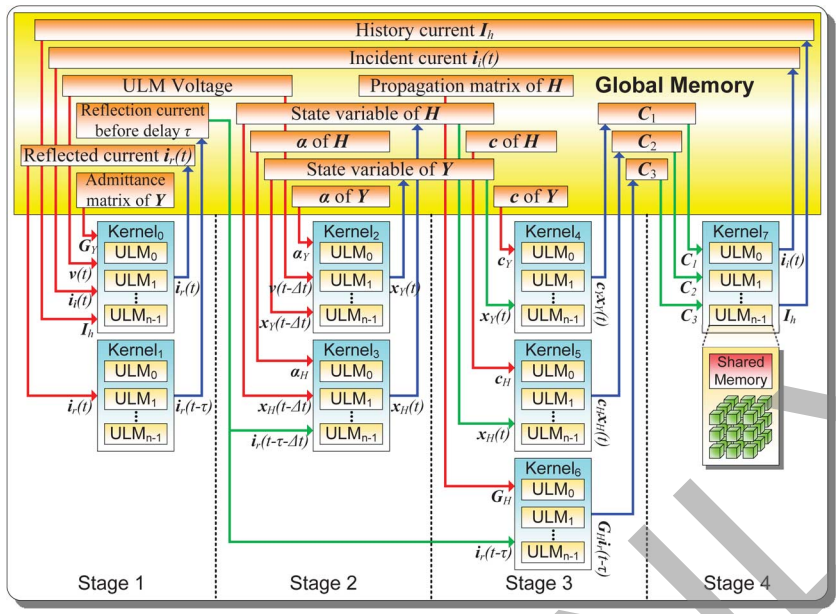


Fig. 7. Massive-thread parallel ULM module.

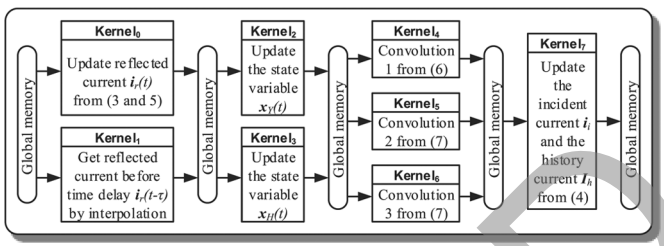


Fig. 8. Kernel operation flow in the ULM module.

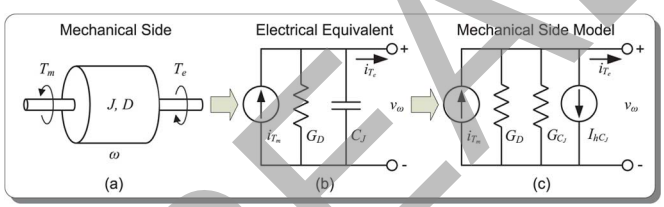


Fig. 9. Electrical model of the mechanical part of UMM.

it provides a unified mathematical framework to model up to 12 types of rotating machines, including asynchronous, synchronous, and dc machines. The electrical part of the UMM includes the armature and field windings. The UMM is allowed to have up to three armature windings (converted to 3 $dq0$ windings), and an unlimited number of windings on the field structure. The mechanical part of the UMM is modeled as an equivalent lumped electric network, where the electromagnetic torque appears as a current source. An alternate representation of the mechanical part as a multimass model (up to a maximum of 6 masses representing various turbine stages) is also possible.

a) Electrical Part: In the UMM used for synchronous machines in this paper, there are three-phase stator armature windings $\{a, b, c\}$, one field winding f, up to 2 damper windings $\{D_1, D_2\}$ on the rotor direct d-axis, and up to three3

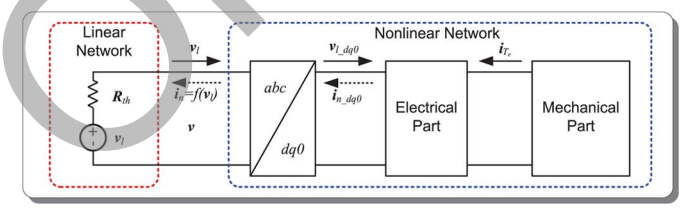


Fig. 10. Interfacing UMM with the network using compensation.

damper windings $\{Q_1, Q_2, Q_3\}$ on the rotor quadrature q-axis. Thus there are a maximum of nine coupled windings whose discretized winding equations are described as

$$\boldsymbol{v}_{dq0}(t) = -\boldsymbol{R}i_{dq0}(t) - \frac{2}{\Delta t}\boldsymbol{\lambda}_{dq0}(t) + \boldsymbol{u}(t) + \boldsymbol{V}_h \qquad (8)$$

where $\boldsymbol{R}$ is the winding resistance matrix, and $\boldsymbol{\lambda}_{dq0}$ is the flux linkage. The history term $\boldsymbol{V}_h$ using Trapezoidal discretization can be expressed as

$$\boldsymbol{V}_h = -\boldsymbol{v}_{dq0} - \boldsymbol{R}i_{dq0} + \frac{2}{\Delta t}\boldsymbol{\lambda}_{dq0} + \boldsymbol{u}. \qquad (9)$$

The Park's transformation orthogonal matrix $\boldsymbol{P}$ links the abc phase domain with the $dq0$ rotating reference domain.

b) Mechanical Part: The dynamics of the rotor, as shown in Fig. 9(a), can be represented as a linear electrical equivalent circuit in the UMM instead of the mass-shaft system, as shown in Fig. 9(b), where the torque T, inertia J, damping D, and rotor speed ω are mapped to the current i_T, capacitance C_J, conductance G_D, and voltage v_ω, respectively. The equivalent differential equation is given as

$$i_{T_m} = C_J\frac{dv_\omega}{dt} + G_D v_\omega + i_{T_e}. \qquad (10)$$

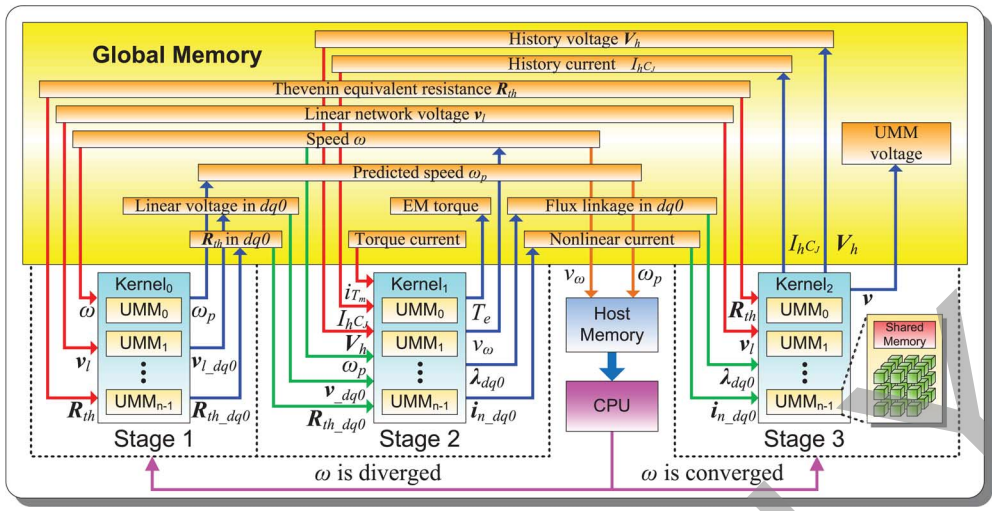


Fig. 11. Massive-thread parallel UMM module.

Discretizing the lumped equivalent capacitance C_J, the mechanical side model is shown in Fig. 9(c). Thus, the equivalent voltage v_ω is expressed as

$$v_\omega(t) = \frac{i_{T_m}(t) - i_{T_e}(t) - I_{hC_J}(t - \Delta t)}{G_D + G_{C_J}} \tag{11}$$

where G_{C_J} is the equivalent conductance and I_{hC_J} is the history current.

Since the UMM is a nonlinear model which connects to the linear network, the compensation method [20] is used to circuit interface it with the EMT network solution. As shown in Fig. 10, the open-circuit node voltage of the nonlinear component $\boldsymbol{v}_l$, which is also the Thévenin equivalent voltage of the linear network, is first solved. Considering $\boldsymbol{v}_l$ as the input to the nonlinear component, the reaction current $\boldsymbol{i}_n$ from the nonlinear system can be calculated by the relational function f between $\boldsymbol{v}_l$ and $\boldsymbol{i}_n$. Injecting $\boldsymbol{i}_n$ into the linear network, the node voltage $\boldsymbol{v}$ of nonlinear components after compensation is given as

$$\boldsymbol{v} = \boldsymbol{v}_l + \boldsymbol{R}_{th}\boldsymbol{i}_n \tag{12}$$

where $\boldsymbol{R}$th is the Thévenin equivalent resistance of the linear network looking into the open port from the nonlinear side.

The electromagnetic torque T_e involving the product of fluxes and currents is calculated iteratively. Once the speed ω has converged, the currents $\boldsymbol{i}_{n\_dq0}$ are transferred back to the phase domain as an incident current $\boldsymbol{i}_n$ from the nonlinear network to the linear network.

2) Massive-Thread Parallel Implementation: As shown in Fig. 11, the designed parallel module for the UMM includes three kernels within three stages. Stage 1 predicts the rotor speed ω_p and transfers the phase-domain inputs into the $dq0$ reference domain. Stage 2 is responsible for the computations of electrical part and mechanical part, and gets the electromagnetic torque T_e, the nonlinear current $\boldsymbol{i}_{n\_dq0}$ in $dq0$, and the equivalent speed voltage v_ω. Before proceeding to Stage 3, the convergence of the rotor speed ω for all UM units is determined by the CPU to avoid the synchronous, efficient, and random memory-access issues arising from the parallel determination, and Stages 1 to 2 are repeated until all UMM units are converged or the maximum number of iterations are reached. Finally, Stage 3 updates the history variables and completes the calculation of the integrated UMM voltage $\boldsymbol{v}$. Similar to the ULM module, each UMM unit occupies a CUDA block running in SIMT, where multiple threads are assigned to handle the vector and matrix operations based on SIMD, according to their dimensions. Shared memory inside the CUDA block is used for critical memory access during the vector and matrix operations.

Fig. 12 shows the operation flow in the kernels of the UMM module. In order to reduce the extra cost for the kernels' switch of the CUDA program, as many as possible tasks are contained in a kernel unless the configuration (threads and memory) of the kernel has to be changed. Inside the kernel$_0$, the rotor speed ω_p is predicted by extrapolation first, then the Park's transformation matrix $\boldsymbol{P}$ is updated to transfer the linear network voltages $\boldsymbol{v}_l$ and Thévenin equivalent resistance $\boldsymbol{R}_{th}$th into the variables $\boldsymbol{v}_{l\_dq0}$ and $\boldsymbol{R}_{th\_dq0}$. The kernel$_1$ first solves the linear system using LU decomposition and forward-backward substitution from (8) to obtain the frame-domain currents $\boldsymbol{i}_{n\_dq0}$. Then, the flux linkages $\boldsymbol{\lambda}_{dq0}$ are updated and, finally, the equivalent speed voltage v_ω is calculated using (11). In kernel$_2$, the reference domain currents $\boldsymbol{i}_{n\_dq0}$ are transferred back to the phase-domain current $\boldsymbol{i}_n$ with the Park's transformation matrix $\boldsymbol{P}^{-1}$ based on the converged rotor speed ω; then, the UMM voltages $\boldsymbol{v}$ are computed from (12) with the linear network voltages $\boldsymbol{v}_l$, Thévenin equivalent resistance $\boldsymbol{R}$th, and the incident currents $\boldsymbol{i}_n$; finally, the history current I_{hC_J} and the history voltages $\boldsymbol{V}_h$ are updated for the next timestep.

IV. Massive-Thread Network Solution

Using the UMM, ULM, and LPE to model a network, the nodal equation for the network is assembled as

$$\boldsymbol{Y}\boldsymbol{v} = \boldsymbol{i} \tag{13}$$

where $\boldsymbol{Y}$ is the admittance matrix, $\boldsymbol{v}$ is the node voltage vector, and $\boldsymbol{i}$ is the vector of nodal current injections. In general, $\boldsymbol{Y}$ is very sparse. For example, for the test system shown in Fig. 13,

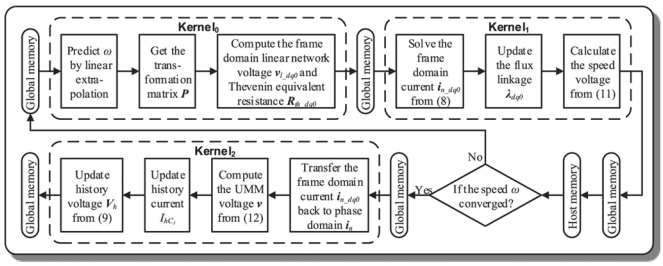


Fig. 12. Kernel operation flow in the UMM module.

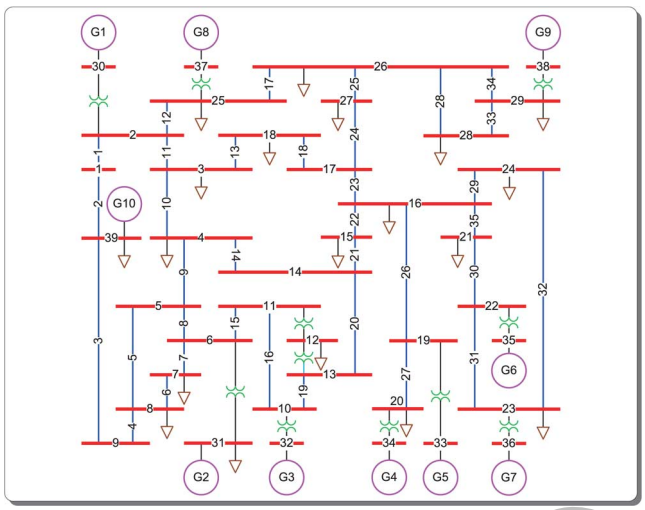


Fig. 13. Single-line diagram of the Scale 1 test power system.

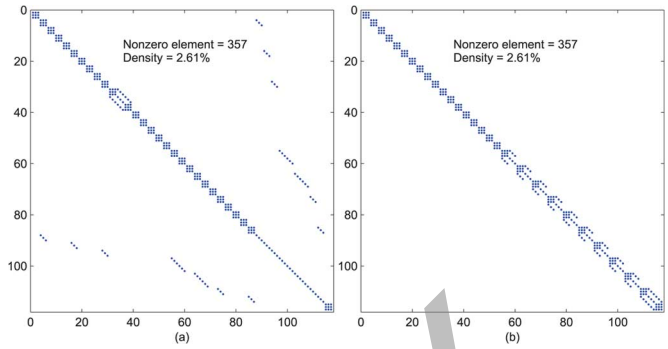


Fig. 14. Pattern of the Y matrix of the IEEE 39-bus system. (a) Before block-node adjustment (BNA). (b) After BNA.

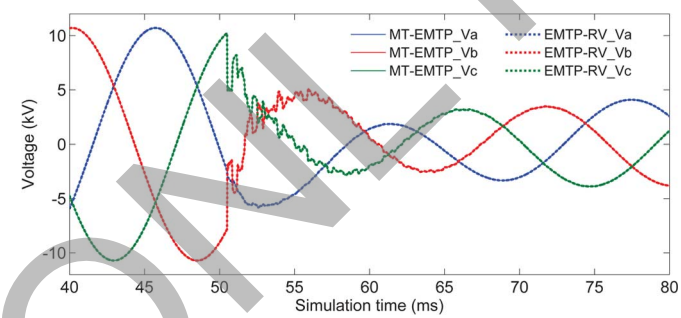


Fig. 15. Comparison of simulation results (three-phase voltages) between MT-EMTP and EMTP-RV at Bus 5 during a three-phase fault at Bus 4.

the original $\boldsymbol{Y}$ is shown in Fig. 14(a). It is 97.39% sparse with 357 nonzero elements. With increasing network size, the admittance matrix becomes even more sparse. It is considerably inefficient to handle a sparse matrix with traditional parallel dense algorithms, and the traditional parallel sparse algorithm is unsuitable for the GPU architecture; therefore, a specific sparse structure, where the nonzero elements are regrouped into diagonal blocks which are decoupled from each other, is proposed using the block-node adjustment (BNA). In the BNA, a graph optimizing algorithm is applied to adjust the node IDs inside each subsystem, which are divided by traveling-wave delays of transmission-line models, into sequential addresses in memory. Since the node IDs created by users are typically random, which do not conform with the rules of the BNA, they are mapped to new node IDs, which are serial and ordered, by the node mapping structure (NMS), where the original IDs are hashed and one-to-one mapping to the new IDs. The minimal perfect hash [27] and integer sorting are applied to avoid string operations so that the complexity is reduced from $O(n^2)$ to $O(n)$.

The $\boldsymbol{Y}$ matrix derived from the NMS is a perfect block diagonal matrix, whose condition number is not affected, as shown in Fig. 14(b), where the number of blocks depends on the number of decoupled systems. Therefore, a large-scale system is divided into n independent smaller subsystems $\boldsymbol{Y}_k \boldsymbol{v}_k = \boldsymbol{i}_k$ $(k = 1, 2, \ldots, n)$, where n is the number of blocks. Only the decoupled blocks in the admittance are stored in the host/device memory, which significantly reduces the pressure of data transfer for large-scale admittance matrices. All small $\boldsymbol{Y}$ matrices are considered as dense matrices, and all subsystems are solved independently by the normal dense algorithm in parallel on the GPU. The linear solver uses LU decomposition and forward-backward substitution, implemented in a CUDA kernel to compute the unknown node voltages $(\boldsymbol{v} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n]')$.

V. Large-Scale EMT Simulation Case Study

The Scale-1 test power system is shown in Fig. 13. It is the modified IEEE 39-bus New England test system. The specifications of the hardware used for the simulation are listed in Table I. The MT-EMTP (64-b code) program was executed on the Fermi GPU, while EMTP-RV (32-bit code) was running on the AMD CPU, both using 64-b double precision floating-point data. The time-domain voltage waveforms at Bus 5 of the test power system (Fig. 13) are shown in Fig. 15, during a three-phase fault event at Bus 4. The fault currents at Bus 4 are shown in Fig. 16. The simulation time is 100 ms and the timestep is 20 μs, and the total simulation steps are 5000. The results from EMTP-RV and MT-EMTP are superimposed in Figs. 15–18, which show close agreement.

The zoomed-in figures (Figs. 17 and 18) also show a close match in the transients.

To evaluate computational efficiency, execution times of test systems of increasing size were recorded. Eight large-scale test cases were created by expanding the original IEEE 39-bus system with detailed modeling of all components. Subsystems (39-bus) were interconnected with the systems around them by two additional transmission lines. The execution times are shown in Table II, which also includes the number of buses

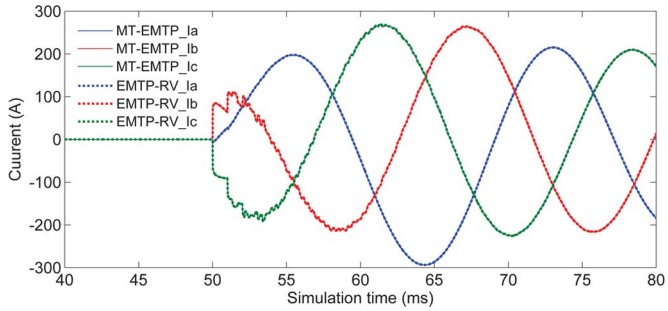


Fig. 16. Comparison of simulation results (three-phase fault currents) between MT-EMTP and EMTP-RV at Bus 4.

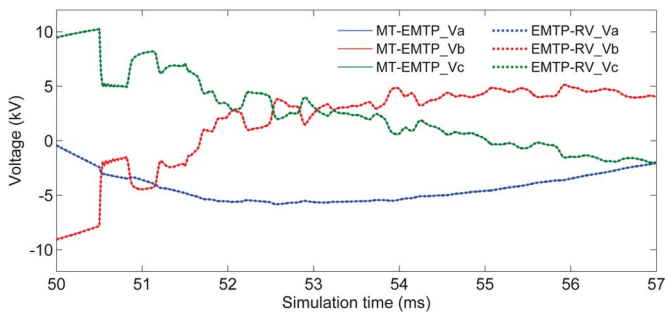


Fig. 17. Zoomed-in view of Fig. 15 from $t = 0.05$ s to $t = 0.057$ s.

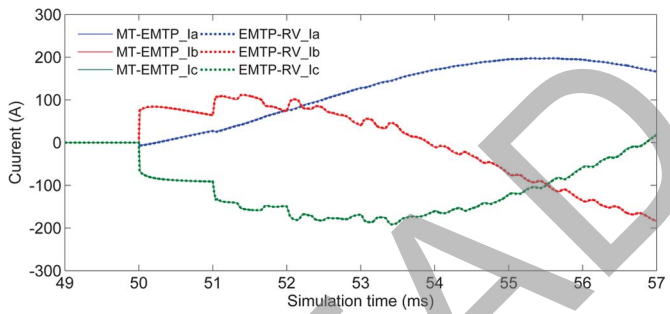


Fig. 18. Zoomed-in view of Fig. 16 from $t = 0.049$ s to $t = 0.056$ s.

TABLE I
HARDWARE SPECIFICATION

GPU		CPU	
Tesla^{TM} C2050 (Fermi)		AMD Phenom^{TM} II 955BE	
Cores	448	Cores	4
Frequency	1.15GHz	Frequency	3.2GHz
Global memory	3GB	System memory	16GB
CUDA Version	4.0	L2 Cache	2MB
CUDA Capability	2.0	L3 Cache	6MB

and devices in the systems. All of the lines in these test cases were modeled using ULM and the uncontrolled machines using UMM. As can been seen, when the system size is relatively small, the speedup is less than 1; however, when the system scale is increased to 63 times of the original IEEE 39-bus system, the achieved speedup is up to 5.63.

Fig. 19 shows the execution time and speedup with increasing system size. It is obvious that the computation time of EMTP-RV follows a high-order complexity $O(n^a)(a > 2)$ with respect to the system scale, since most vector and matrix operations have high-order complexity $O(n^2)$ and $O(n^3)$ in

TABLE II
COMPARISON OF EXECUTION TIME FOR VARIOUS SYSTEM SIZES BETWEEN EMTP-RV AND GPU-BASED MT-EMTP FOR SIMULATION DURATION 100 ms WITH TIMESTEP 20 μs

System structure (3-phase)					Execution time (s)		Speedup
Scale	Buses	Devices			EMTP-RV	MT-EMTP	
		LPEs	ULMs	UMMs			
1	40	31	35	10	0.967	1.837	0.53
2	79	61	72	20	2.012	2.350	0.86
4	157	121	148	40	4.118	3.304	1.25
8	313	241	300	80	9.625	5.345	1.80
16	625	481	608	160	26.988	9.116	2.96
32	1249	961	1224	320	65.567	16.895	3.88
48	1873	1441	1840	480	115.724	23.319	4.96
63	2458	1891	2425	630	168.502	29.946	5.63

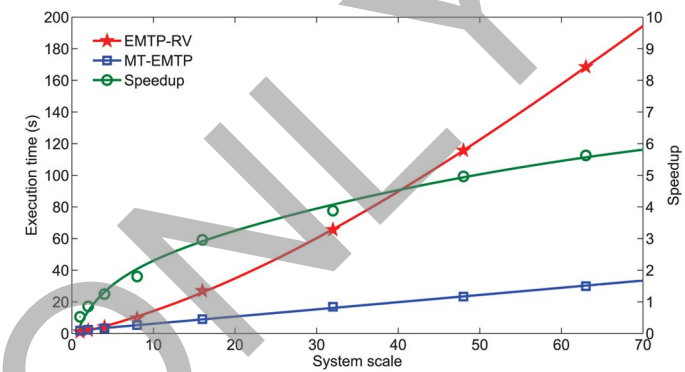


Fig. 19. Execution time and speedup with respect to the scale of test systems in EMTP-RV and the GPU-based MT-EMTP program.

serial CPU algorithms. The execution time of the proposed MT-EMTP program, however, only increases linearly with first-order complexity $O(n)$, derived from SIMD-based parallel programming. Thanks to the complexity order reduction, a GPU-based EMT simulator is always faster than the conventional CPU-based simulator when the scale of the test case is large enough. Therefore, the speedup can be expected to increase without saturation for increasing system sizes. Larger systems (greater than Scale 63) could also be tested on MT-EMTP, but the EMTP-RV licence only allowed a maximum of 5000 devices. Note that in a commercial and industrial program, such as EMTP-RV, there are many input and output activities, and a large collection of models/codes that require extra processing time. Nevertheless, this experiment clearly demonstrates the advantage of parallel massive-thread computation in accelerating EMT simulation.

VI. CONCLUSION

The GPU with its massive-core architecture shows promise for higher computational performance, provided the system size and the data throughput requirements are large. Detailed EMT simulation of the large-scale system is therefore ideally suited to exploit GPU technology for reducing computational burden. This paper proposed a parallel massive-thread module for linear passive element, transmission line, and electrical machine for implementing the MT-EMTP program on the GPU, a node-mapping structure is proposed for an efficient sparse network solution. Large-scale test cases are used to evaluate the performance of the MT-EMTP program in comparison with the commercial EMTP-RV software. With lower order complexity,

the massive-thread parallel implementation shows substantial speedup under the same accuracy and precision. Finally, the proposed methods, algorithm, and data structure also can be applied to the multithread computing system, which is also pervasive these days as a mainstream CPU architecture.

APPENDIX A

The parameters for the test power system in Fig. 13 are given below:

1) ULM transmission-line (Line1–Line35) parameters: three conductors, resistance: 0.0583/km, diameter: 3.105 cm, line length: 50 km (line 5, 6, 7, 8, 15, 16, 18, 19, 23, 27, 29, 30, 31, 35); 150 km (line 2, 3, 4, 9, 10, 11, 13, 14, 20, 21, 22, 24, 25, 26, 32, 33); and 500 km (line 1, 12, 17, 28, 34). $\boldsymbol{Y}$ and $\boldsymbol{H}$ are 3×3 matrices, whose elements are approximated with ninth-order rational functions. Line geometry: flat horizontal phase spacing; horizontal distance between $\text{adjacent phases} = 4.87$ m; vertical distance: phases a to ground, c to ground $= 30$ m, phase b to ground $= 28$ m, and shield wire to tower arm $= 6$ m.
2) UMM synchronous machine (G1–G10) parameters: 1000 MVA, 22 kV, Y-connected, field current: 2494 A, 2 poles, 60 Hz, moment of inertia: 5.628e4 kg·m^2/rad and damping: 6.780e3 kg·m/s/rad.
3) Loads and transformer parameters: load parameter: $\mathrm{R} = 500\,\Omega$, $\mathrm{L} = 0.05$ H, $\mathrm{C} = 1\,\mu$F and transformer leakage impedance $\mathrm{R} = 0.5\,\Omega$, $\mathrm{L} = 0.03$ H.

REFERENCES

[1] J. Mahseredjian, S. Dennetière, L. Dubé, B. Khodabakhchian, and L. Gérin-Lajoie, "On a new approach for the simulation of transients in power systems," presented at the Int. Conf. Power Syst. Transients, Montreal, QC, Canada, Jun. 19–23, 2005, IPST05-139.

[2] Manitoba HVDC Research Centre, in *EMTDC User's Guide*, Winnipeg, MB, Canada, 2003.

[3] *"Alternative Transients Program (ATP) Rule Book,"* CAN/AM EMTP Users Group, 2000.

[4] J. Mahseredjian, V. Dinavahi, and J. A. Martinez, "Simulation tools for electromagnetic transients in power systems: Overview and challenges," *IEEE Trans. Power Del.*, vol. 24, no. 3, pp. 1657–1669, Jul. 2009.

[5] L. Gérin-Lajoie and J. Mahseredjian, "Simulation of an extra large network in EMTP: From electromagnetic to electromechanical transients," presented at the Int. Conf. Power Syst. Transients, Kyoto, Japan, Jun. 3–6, 2009.

[6] NVIDIA Corporation, NVIDIAs Next Generation CUDA Compute Architecture: Fermi, 2009.

[7] D. Blythe, "Rise of the graphics processor," *Proc. IEEE*, vol. 96, no. 5, pp. 761–778, May 2008.

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

[9] *"NVIDIA CUDA C Programming Guide Version 4.0,"* NVIDIA Corp., May 6, 2011.

[10] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units," in *Proc. IEEE Power Tech.*, Jul. 1–5, 2007, pp. 731–736.

[11] J. E. Tate and T. J. Overbye, "Contouring for power systems using graphical processing units," in *Proc. 41st Annu. Int. Conf. Syst. Sci.*, HI, USA, Jan. 7–10, 2008, p. 168.

[12] V. Jalili-Marandi and V. Dinavahi, "Large-scale transient stability simulation on graphics processing units," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Jul. 26-30, 2009, pp. 1–6.

[13] N. Garcia, "Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Jul. 25–29, 2010, pp. 1–4.

[14] V. Jalili-Marandi and V. Dinavahi, "SIMD-based large-scale transient stability simulation on the graphics processing unit," *IEEE Trans. Power Syst.*, vol. 25, no. 3, pp. 1589–1599, Aug. 2010.

[15] J. Singh and I. Aruni, "Accelerating power flow studies on graphics processing unit," in *Proc. IEEE Ann. India Conf.* , Dec. 17–19, 2010, pp. 1–5.

[16] C. Vilacha, J. C. Moreira, E. Miguez, and A. F. Otero, "Massive Jacobi power flow based on SIMD-processor," in *Proc. 10th Int. Conf. Envir. Elect. Eng.*, May 8–11, 2011, pp. 1–4.

[17] J. K. Debnath, W. Fung, M. A. Gole, and S. Filizadeh, "Simulation of large-scale electrical power networks on graphics processing units," in *Proc. IEEE Elect. Power Energy Conf.*, Oct. 3–5, 2011, pp. 199–204.

[18] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 7, pp. 1255–1266, Jul. 2012.

[19] H. W. Dommel, "Digital computer solution of electromagnetic transients in single and multiphase networks," *IEEE Trans. Power App. Syst.*, vol. PAS-88, no. 4, pp. 388–399, Apr. 1969.

[20] H. W. Dommel, "Nonlinear and time-varying elements in digital simulation of electromagnetic transients," *IEEE Trans. Power App. Syst.*, vol. PAS-90, no. 4, pp. 2561–2567, Jun. 1971.

[21] H. W. Dommel, *EMTP Theory Book*. Portland, OR, USA: Bonneville Power Admin., 1984.

[22] J. R. Marti, "Accurate modeling of frequency-dependent transmission lines in electromagnetic transients simulations," *IEEE Trans. Power App. Syst.*, vol. PAS-101, no. 1, pp. 147–157, Jan. 1982.

[23] H. K. Lauw and W. S. Meyer, "Universal machine modeling for the representation of rotating electric machinery in an electromagnetic transients program," *IEEE Trans. Power App. Syst.*, vol. PAS-101, no. 6, pp. 1342–1350, Jun. 1982.

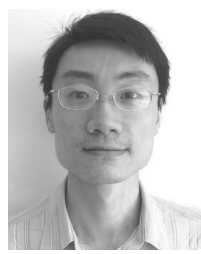
[24] A. Morched, B. Gustavsen, and M. Tartibi, "A universal model for accurate calculation of electromagnetic transients on overhead lines and underground cables," *IEEE Trans. Power Del.*, vol. 14, no. 3, pp. 1032–1038, Jul. 1999.

[25] H. K. Lauw, "Interfacing for universal multi-machine system modeling in an electromagnetic transients program," *IEEE Trans. Power App. Syst.*, vol. PAS-104, no. 9, pp. 2367–2373, Sep. 1985.

[26] B. Gustavsen and A. Semlyen, "Simulation of transmission line transients using vector fitting and modal decomposition," *IEEE Trans. Power Del.*, vol. 13, no. 2, pp. 605–614, Apr. 1998.

[27] R. J. Cichelli, "Minimal perfect hash functions made simple," *Commun. ACM*, vol. 23, no. 1, pp. 17–19, Jan. 1980.

[28] J. R. Marti and J. Lin, "Suppression of numerical oscillations in the EMTP," *IEEE Trans. Power Syst.*, vol. 4, no. 2, pp. 739–747, May 1989.



Zhiyin Zhou (S'12) received the M.Sc. degree in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2012 and is currently pursuing the Ph.D. degree in electrical and computer engineering at the University of Alberta.

His research interests include large-scale parallel and distributed computing, massive-thread parallel programming, power system simulation, and electromagnetic transient studies.

Venkata Dinavahi (SM'08) is a Professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada.

His research interests include real-time simulation of power systems and power-electronic systems, large-scale system simulation, and parallel and distributed computing.