Large-Scale Nonlinear Device-Level Power Electronic Circuit Simulation on Massively Parallel Graphics Processing Architectures

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

***Abstract*—Device-level power electronic circuit simulation is so cumbersome that engineers are forced to make model simplification or reduce circuit size to obtain a reasonable execution time for repeated simulation runs. This paper proposes a massive-thread parallel simulation of large-scale power electronic circuits employing device-level modeling on the graphics processors (GPUs) to obtain higher data throughput and lower execution times. Parallel massive-thread modules are proposed for the nonlinear physics-based insulated gate bipolar transistor and power diode components. The nonlinear solution algorithm comprised of Newton–Raphson iterations and partial LU decomposition is fully parallelized on the GPU. Furthermore, the commonly used behavioral model with reduced computational complexity is also employed to represent the switches. The developed simulation codes are used to run large-scale test cases of the modular multilevel converter (MMC) system. The accuracy and efficiency of the GPU-based parallel simulation are compared with sequential CPU-based codes and the SaberRD program to show the advantages of the parallelized simulation; execution time speedups of 15 times and 70 times are reported for the MMC system using the nonlinear physics-based modeling and behavior-based modeling, respectively.**

***Index Terms*—Device-level modeling, graphics processors, insulated gate bipolar transistor (IGBT), large-scale circuits, massive-thread, modeling, numerical analysis, parallel algorithms, parallel architectures, power diode, power electronics, simulation software.**

I. Introduction

COMPUTATIONAL speed is an overriding concern in large-scale power electronic circuit simulation using device-level circuit simulators. Modeling complex systems composed of power electronic subsystems, such as HVDC grids, transportations, renewable energy, and smart grid technologies, can often be very challenging as the system modeler is faced with a difficult compromise between system size, modeling complexity, and simulation duration to obtain a reasonable execution time for the application. Computational bottlenecks arise during repeated nonlinear transient simulations, which are quite frequently required in many studies that include but are not limited to the following:

1) robust design of power electronic systems involving optimization to fine-tune parameters at the circuit and component levels requiring several design iterations and hundreds of simulation runs;
2) in addition to transient analysis, simulation-based statistical and sensitivity analysis of the system hierarchy to reduce design costs and time;
3) comprehensive fault analysis of systems using a matrix of faults representing possible device/component failures requiring multiple simulation runs to evaluate system performance and improve reliability;
4) data visualization and analysis of large sets of postsimulation results to extract meaningful system performance indices;
5) detailed modeling of complex mixed-signal and multidomain subsystems with widely different time constants, for e.g., electronic, electrical, magnetic, thermal, and hydraulic systems.

There are a variety of device-level simulators available for power electronic circuit simulation, both commercial and noncommercial [1]–[3]. A partial listing of such tools include SaberRD, Orcad, PSIM, PLECS, LTSpice, PECS, PETS, DesignLab, etc. All of these simulation tools provide a plethora of models for semiconductor devices such as diodes, bipolar junction transistor (BJTs), JFETs, MOSFETs and insulated gate bipolar transistor (IGBTs) and fundamental circuit components such as linear/nonlinear resistors, capacitors, inductors, and independent/dependent voltage and current sources. These software tools are also capable of performing an assortment of studies such as nonlinear dc, transient, linear ac (small signal), and Monte Carlo analyses. Furthermore, since many design projects may include analog, digital, and mixed-signal simulations, most of these tools either possess native mixed-signal capabilities or they provide cosimulation interfaces to leverage the features of an external toolset [4]–[7]. A key attribute shared by currently available simulation tools in terms of program execution is that they are single-thread programs designed to run sequentially on the CPU. Although some modifications have been made for distributed processing on multiple CPUs, such as the distributed iterative analysis in SaberRD [8], the actual execution of program code on individual CPUs is still sequential. Therefore, the

Manuscript received February 15, 2017; revised April 20, 2017; accepted July 6, 2017. Date of publication July 11, 2017; date of current version February 22, 2018. This work was supported by the Natural Science and Engineering Research Council of Canada. Recommended for publication by Associate Editor J. A. Oliver. *(Corresponding author: Venkata Dinavahi.)*

S. Yan is with Tesla, Inc., Shanghai 201108, China (e-mail: shyan@tesla.com).

Z. Zhou and V. Dinavahi 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/TPEL.2017.2725239

attained *task parallelism* is coarse grained at best. The resulting computational efficiency of device-level circuit simulators was primarily derived from an increase in CPU clock speed, which until the mid-2000s could be relied upon to provide the necessary acceleration. However, computer chip manufactures no longer rely on clock speed for higher processing power; they have implemented multicore CPU and many-core GPU architectures to increase chip performance. While multiple thread concepts on CPUs such as hyperthreading were introduced early on, they were hardly taken advantage of by circuit simulators mainly due to the cumbersome task of rewriting the program code to enable multiple threads of execution.

Model order reduction is the usual course for improving computational speed in device-level circuit simulators. Model simplifications include circuit size reduction and using averaged or linearized models for the switching devices to observe only the system-level behavior. However, evaluating the system's comprehensive behavior entails maintaining many different models of varying size and complexity on different simulation tools. It would be far more effective if the same simulation tool could efficiently show both the system-level and device-level results over long time frames. Taking IGBT as an example, there are several models ranging from detailed physics-based model to ideal switch model. Hefner brought up the first complete analytical physics-based model available for a device-level circuit simulator, which is implemented in SaberRD [9], [10]. As the most common voltage source converter for HVdc applications, modular multilevel converter (MMC) is normally simulated using simplified IGBT and diode models because of its complex structure consisting of a series of submodules (SM); device-level details of IGBT and diode cannot be observed using simplified system-level models of the MMC.

Since GPU offers a massively parallel architecture composed of thousands of cores grouped into streaming multiprocessors though its clock frequency is relevantly lower than CPU, it has a higher compute power and throughput in floating point calculations than traditional CPUs, as compared in Table III. The attained *data parallelism* is fine-grained, which conforms the single instruction, multiple data (SIMD) format; therefore, to fully exploit GPU acceleration, the device models and the numerical algorithms have to be rewritten into the SIMD format [11]. Mature application programming interfaces are available for SIMD abstraction such as CUDA, DirectCompute, and OpenCL. The user develops C/C++ code interlaced with special constructs and functions to access the parallel cores and distributed memories on the GPU. Furthermore, optimized numerical libraries such as CUBLAS [12] and CUFFT [13] are available for linear solvers and data processing. GPU-based massively parallel processing has been used worldwide for myriad applications, such as computational fluid dynamics, life sciences, medical imaging, game physics, seismic simulations, etc., and impressive acceleration has been reported [14]–[17]. For power system computation, GPUs have been used for various studies, such as transient stability simulation [18], electromagnetic transient simulation [19], dynamic state estimation [20], [21], ionized field computation [22], and power flow calculation [23].

TABLE I
GPU SPECIFICATIONS

Chip	GK110 (Kepler)	GP104 (Pascal)
Fabrication	28 nm	16 nm
Number of single precision cores	2880	2560
Memory bandwidth	336 GB/s	320 GB/s
Memory size	6 GB	8 GB
Memory type	GDDR5	GDDR5X
Core clock	837 MHz	1607 MHz

In this paper, a massively parallel simulation of large-scale power electronic circuits using nonlinear physics-based device-level models on the GPU architecture is proposed. The massive-thread implementation of physics-based IGBT and power diode utilize the Hefner's IGBT model and Lauritzen's diode model [24]. The numerical solution modules including the Newton–Raphson iterative method, matrix equation solution using partial LU decomposition are implemented in the massive-thread framework. Based on the MMC circuit characteristic, a mathematical method is proposed to decompose the semiblock Jacobian matrix. In addition, a variable time-stepping scheme using the predictor–corrector method is adopted to increase simulation efficiency. The GPU-based massive-thread simulation is compared with SaberRD simulator as well as a complete CPU implementation, in terms of accuracy and computational efficiency.

This paper is organized as follows. Section II briefly describes the GPU hardware architecture and CUDA abstraction for parallel computation. Section III describes the details of the developed massive-thread parallel modules for the IGBT, power diode, and the numerical algorithms. Section IV shows the case studies of MMC with simulation results and discussion. Finally, Section V gives the conclusion of this paper.

II. BACKGROUND ON GPU ARCHITECTURE AND PROGRAMMING INTERFACE

Two GPU architectures, NVIDIA Kepler GK110 (2012) and Pascal GP104 (2016), whose specifications are given in Table I, are used to implement the massively parallel power electronic circuit simulation codes. In hybrid computational systems, CPU and GPU cooperate as *host* and *device*, respectively. The host sends all essential instructions and data to the device through PCIe 3.0 $\times$ 16 interface up to 15.754 GB/s. Instructions are distributed through GigaThread interface to each streaming multiprocessor and data are transferred to global memory on the GPU board. Every 32 threads in the streaming multiprocessor are distributed into one *warp* as an execution unit, which operate simultaneously, while other threads are parallelized by the pipeline automatically. Finally, results saved in global memory are transferred back to host through PCIe 3.0 bus again.

Growing from NVIDIA's Fermi architecture, the Kepler architecture has 15 streaming multiprocessors with registers, caches, and shared memory, shown in Fig. 1(a) [25]. Each multiprocessor contains 192 CUDA cores, which is 3$\times$ that

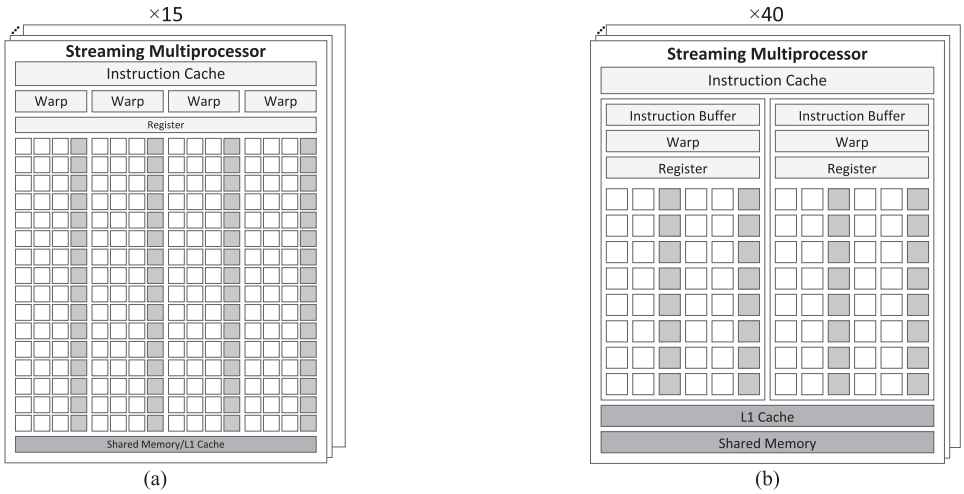


Fig. 1. Stream microprocessor structure of NVIDIA Kepler and Pascal architectures. (a) Kepler. (b) Pascal.

of its predecessor, Fermi. A CUDA core is an instruction executing floating point operation in one thread. As the successor of NVIDIA Kepler architecture, the Pascal architecture contains more powerful, programmable, and power-efficient stream multiprocessors, shown in Fig. 1(b) [26]. Although each stream multiprocessor only has 64 CUDA cores, there are 40 stream multiprocessors in total for the Pascal architecture with much higher clock frequency, larger memory size, and wider bandwidth, as listed in Table I, thanks to its more advantageous 16 nm fabrication. There are several types of memories on board that includes global memory, shared memory, and registers.

The global memory is larger with access to the entire device but has a higher latency; the shared memory can be accessed by all cores inside the streaming multiprocessor with low latency; and there are a few registers inside a streaming multiprocessor that are the fastest. In the Kepler architecture, shared memory can be configured in the sizes of 16, 32, or 48 kB, which shares the 64 kB on-chip memory with L1 Cache, while in the Pascal architecture, the amount of shared memory is up to 96 kB. The memory management, including allocation and organization, is the key to programming efficiency. CUDA offers both a platform and programming model for NVIDIA's GPU [27]. The function involving data parallelism is referred to as a *kernel*, which organizes massive *threads* into *blocks* inside a *grid*. All threads inside a kernel must be synchronized before the end of execution. The *block*-level barrier, a synchronization point ensures that all threads inside a block have reached the command line and are ready for next instruction. The *device*-level synchronization barrier synchronizes all thread procedures inside a grid before the next kernel execution.

III. Massive-Thread Parallel Modules for Power Electronic Circuit Simulation

A. Nonlinear Power Diode

1) Model Formulation: Detailed device-level modeling of power diodes covers a wide range of circuit operating conditions since it includes equations for drift and diffusion of electrons and holes. However, different from conventional detailed models, which are too complicated to simulate, this paper employs a simplified physics-based model containing p-i-n structure suitable for power diode operating condition of high voltage and fast switching [28].

The physical structure of a power diode is shown in Fig. 2(b). The reverse recovery happens when turning off a forward conducting diode rapidly, as described by the following equations:

$$i_R(t) = \frac{q_E(t) - q_M(t)}{T_M} \tag{1}$$

$$0 = \frac{dq_M(t)}{dt} + \frac{q_M(t)}{\tau} - \frac{q_E(t) - q_M(t)}{T_M} \tag{2}$$

$$q_E(t) = I_S \tau (e^{\frac{v_E(t)}{V_T}} - 1) \tag{3}$$

where $i_R(t)$ is the diffusion current in *i*-region, $q_E(t)$ represents charge variable in the junction area, $q_M(t)$ represents charge variable in the middle of *i*-region, T_M is the diffusion transit time across i-region, τ is the lifetime of recombination, I_S is the diode saturation current constant, v_E is the junction voltage, and V_T is the thermal voltage constant. The voltage drop across i-region $v_M(t)$ is described as

$$v_M(t) = \frac{V_T T_M i(t)}{q_M(t)}. \tag{4}$$

The voltage across diode $v(t)$ is expressed as

$$v(t) = 2v_M(t) + 2v_E(t) + R_S \left[i_E(t) + \frac{dq_J(t)}{dt} \right] \tag{5}$$

where R_S is the contact resistance presented as an internal resistance and the charge of junction capacitance in the capacitance C_J is given as

$$q_J(t) = \int C_J(t) d(2v_E). \tag{6}$$

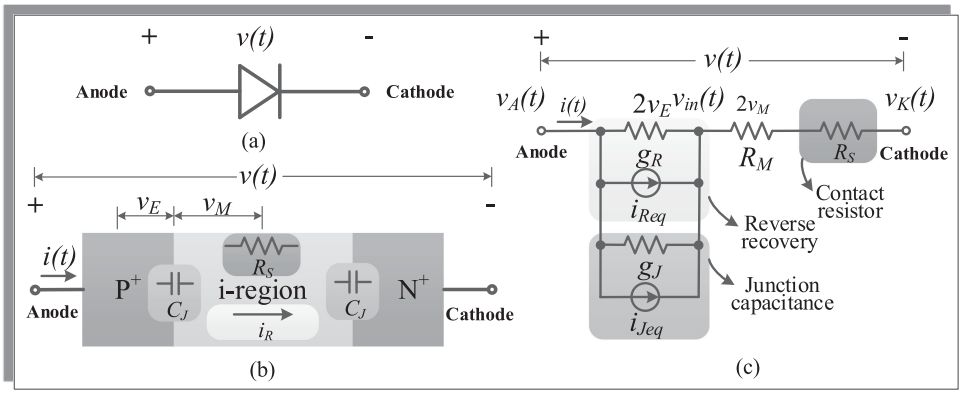


Fig. 2. (a) Power diode symbol. (b) Physical structure of power diode. (c) Discretized and linearized equivalent circuit of power diode (Diode-DLE).

The expression of junction capacitance $C_J(t)$ is given as follows:

$$C_J(t)=\begin{cases}\dfrac{C_{J0}}{\left(1-\frac{2v_E(t)}{\phi_B}\right)^m}, & v_E<\dfrac{\phi_B}{4}\\[2ex] \left[\dfrac{2^{m+2}mv_E(t)}{\phi_B}-(m-1)2^m\right]C_{J0}, & v_E\geq\dfrac{\phi_B}{4}\end{cases}, \quad (7)$$

where C_{J0} is the zero-biased junction capacitance, ϕ_B is the built-in potential, and m is the junction grading coefficient.

2) Model Discretization and Linearization: After discretization by trapezoidal rule, the differential term $dq_M(t)/dt$ in (2) is expressed as

$$q_M=\frac{\Delta t\cdot q_E(t)}{2T_M(1+\frac{k_1\Delta t}{2})}+\frac{q_{\rm hist}(t-\Delta t)}{1+\frac{k_1\Delta t}{2}} \quad (8)$$

where the history term is given as

$$q_{\rm hist}(t-\Delta t)=\frac{\Delta t}{2T_M}q_E(t-\Delta t)-\frac{k_1\Delta t}{2}q_M(t-\Delta t). \quad (9)$$

The equivalent reverse recovery current $i_{R\rm eq}$, as shown in Fig. 2(c), is given as

$$i_{R\rm eq}=k_2I_S\tau(e^{\frac{v_E(t)}{v_T}}-1)-\frac{q_{\rm hist}(t-\Delta t)}{T_M(1+\frac{k_1\Delta t}{2})}-2v_E(t)g_R \quad (10)$$

where g_R is the dynamic conductance defined as

$$g_R=\frac{1}{2v_T}k_2I_S\tau e^{\frac{v_E(t)}{V_T}}. \quad (11)$$

Similarly, the equivalent junction capacitance current $i_{J\rm eq}$ is obtained as

$$i_{J\rm eq}=i_J(t)-2v_E(t)g_J \quad (12)$$

where the equivalent junction conductance g_J is given as

$$g_J=\frac{2}{\Delta t}C_J(t). \quad (13)$$

The discretized and linearized system (Diode-DLE) shown in Fig. 2(c) can be obtained as follows:

$$\boldsymbol{G}^{\rm Diode}\cdot\boldsymbol{V}^{\rm Diode}=\boldsymbol{I}_{\rm eq}^{\rm Diode} \quad (14)$$

Algorithm 1: Diode Implementation.

procedure PHYSICS BASED POWER DIODE MODULE
- Reverse Recovery:
- Calculate $q_E(t)$ from $v_E(t)$
- Update $q_{\rm hist}(t)$ (9)
- Calculate $i_{R\rm eq}$ (10) and g_R (11) ▷ Kernel$_0$
- Junction capacitance:
- Update $C_J(t)$ (7)
- Calculate $i_{J\rm eq}$ (12) and g_J (13) ▷ Kernel$_0$
- Complete module:
- Solve nonlinear system ▷ Kernel$_1$
- Update $v_E(t)$
- **if** $v_E(t)$ not converged **then**
  - **go to** Reverse Recovery:
- **else**
  - Output solution

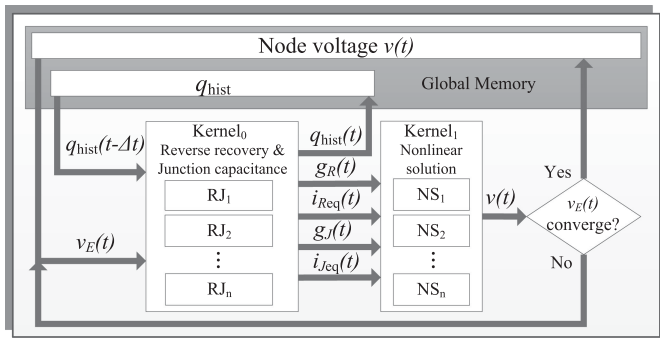


Fig. 3. Massive-thread parallel implementation of power diode.

where

$$\boldsymbol{G}^{\rm Diode}=\begin{bmatrix} g_R+g_J & -g_R-g_J & 0\\ -g_R-g_J & g_R+g_J+\frac{1}{R_M+R_S} & -\frac{1}{R_M+R_S}\\ 0 & -\frac{1}{R_M+R_S} & \frac{1}{R_M+R_S}\end{bmatrix} \quad (15)$$

$$\boldsymbol{V}^{\rm Diode}=[v_A,\quad v_{in},\quad v_K]^T \text{ and} \quad (16)$$

$$\boldsymbol{I}_{\rm eq}^{\rm Diode}=[-i_{R\rm eq}-i_{J\rm eq},\quad i_{R\rm eq}+i_{J\rm eq}\quad ,0]^T. \quad (17)$$

Applying the massive-thread parallel Newton–Raphson method, which is detailed in Section III-D, the next iterate values $\boldsymbol{V}^{{\rm Diode}(n+1)}$ can be updated by previous nth iterate values until the solution is converged.

3) Parallel Massive-Thread Mapping: As shown in Fig. 3 and described in Algorithm 1, there are two kernels in the massive-thread parallel diode model. The dynamic conductance

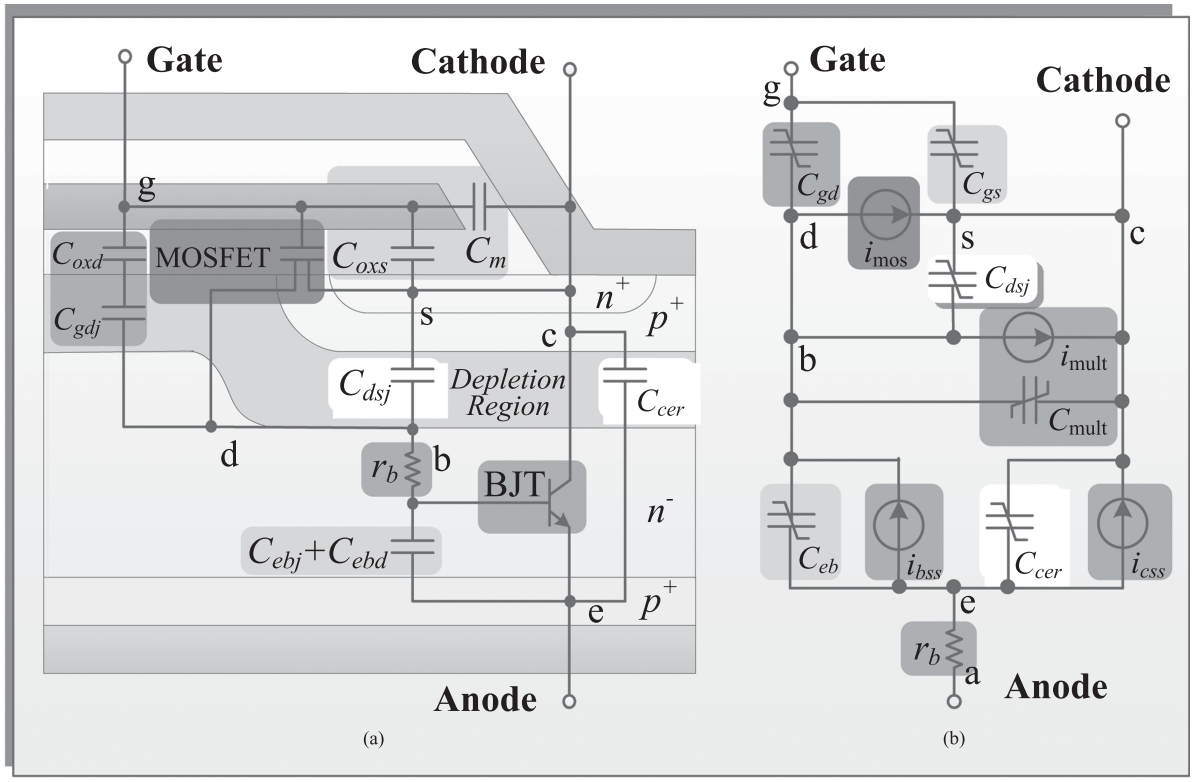


Fig. 4. (a) Phenomenological structure of IGBT. (b) Analog equivalent circuit of IGBT (IGBT-AE).

g_R and equivalent reverse recovery current $i_{R\text{eq}}$ are updated by the junction voltage v_E, and the equivalent conductance g_J and current source $i_{J\text{eq}}$ are updated by the junction capacitance C_J in reverse recovery and junction capacitance units (RJs) of Kernel$_0$. The nonlinear system is solved using the massive-thread parallel Newton–Raphson iteration method in nonlinear solution units (NSs) of Kernel$_1$. The convergence of $v_E(t)$ is checked and it determines whether the process will move to the next time step.

B. Nonlinear Physics-Based IGBT

1) Model Formulation: Based on Hefner's physics-based model [10], the IGBT is described as the combination of a bipolar transistor and a MOSFET. Since these internal devices are differently structured from standard microelectronic devices, a regional approach is adopted to identify the phenomenological circuit of IGBT, as shown in Fig. 4(a). An analog equivalent circuit, shown in Fig. 4(b), makes it possible to implement the model in circuit simulators by replacing the BJT with base and collector current sources and MOSFET with a current source, which represents the currents between each of the terminals and internal nodes in terms of nonlinear functions.

a) Currents: The steady-state collector current i_{css} of BJT is formulated as

$$i_{css} = \frac{i_T}{1+b} + \frac{4bD_pQ}{(1+b)W^2} \tag{18}$$

where b is the ambipolar mobility ratio, D_p is the hole diffusivity, W is the quasi-neutral base width, Q is the instantaneous excess-carrier base charge, and the anode current i_T is shown as follows:

$$i_T = \frac{v_{ae}}{r_b}. \tag{19}$$

The base resistance r_b in (19) is expressed as

$$r_b = \begin{cases} \frac{W}{q\mu_n AN_B} & v_{eb} \le 0 \\ \frac{W}{q\mu_{\text{eff}} An_{\text{eff}}} & v_{eb} > 0 \end{cases} \tag{20}$$

where μ_n and μ_{eff} stand for electron mobility and effective mobility, n_{eff} is the effective doping concentration, q is the electron charge, N_B is the base doping concentration, and A is the device active area. The steady-state base current i_{bss} is caused by the decay of excess base charge of recombination in the base and electron injection in the emitter, and is expressed as

$$i_{bss} = \frac{Q}{\tau_{\text{HL}}} + \frac{4Q^2N_{scl}^2 i_{sne}}{Q_B^2 n_i^2} \tag{21}$$

where τ_{HL} is the base high-level lifetime, N_{scl} is the collector–base space concentration, i_{sne} is the emitter electron saturation current, n_i is the intrinsic carrier concentration, and Q_B represents the background mobile carrier base charge. The MOSFET channel current i_{mos} is expressed as

$$i_{\text{mos}} = \begin{cases} 0, & v_{gs} < v_T \\ K_p(v_{gs}-v_T)v_{ds} - \frac{K_p v_{ds}^2}{2} & v_{ds} \le v_{gs} - v_T \\ \frac{K_p(v_{gs}-v_T)^2}{2}, & v_{ds} > v_{gs} - v_T \end{cases} \tag{22}$$

where K_p is the MOSFET transconductance parameter, v_{gs} is the gain-source voltage, and v_T is the MOSFET channel threshold voltage [29]. In addition, due to thermal generation in the depletion region and carrier multiplication, which is a key factor to determine the avalanche breakdown voltage and the leakage current, the avalanche multiplication current $i_{\rm mult}$, shown in Fig. 4(b), is given as

$$i_{\rm mult} = (M-1)(i_{\rm mos} + i_{css} + i_{c\_cer}) + M i_{\rm gen} \tag{23}$$

where M stands for the avalanche multiplication factor.

b) Charges and capacitances: The gate–source capacitance C_{gs} in the analog model is a constant, and its charge Q_{gs} is given as

$$Q_{gs} = C_{gs} v_{gs}. \tag{24}$$

The gate–drain capacitance C_{gd} is expressed as

$$C_{gd} = \begin{cases} C_{oxd}, & v_{ds} \le v_{gs} - v_{Td} \\ \frac{C_{gdj} C_{oxd}}{C_{gdj} + C_{oxd}} & v_{ds} > v_{gs} - v_{Td} \end{cases} \tag{25}$$

where v_{Td} is the gate–drain overlap depletion threshold voltage, C_{oxd} is the gate–drain capacitance. The gate–drain overlap depletion capacitance C_{gdj} is given as

$$C_{gdj} = \frac{A_{gd}\epsilon_{si}}{W_{gdj}} \tag{26}$$

where A_{gd} is the gate–drain overlap area, ϵ_{si} is the silicon dielectric constant, and W_{gdj} is the gate–drain overlap depletion width. The charge of C_{gd} has the expression as (27) shown at the bottom of this page. Similarly, the drain–source depletion capacitance C_{dsj}, related to the active area $(A - A_{gd})$ and drain–source depletion width W_{dsj}, is given as

$$C_{dsj} = \frac{(A - A_{gd})\epsilon_{si}}{W_{dsj}} \tag{28}$$

and its charge Q_{ds} is expressed as

$$Q_{ds} = A_{ds}\sqrt{2\epsilon_{si}(v_{ds} + 0.6) q N_{scl}}. \tag{29}$$

The emitter–base capacitance C_{eb} is solved from $\frac{\partial Q_{eb}}{\partial V_{eb}}$ as

$$C_{eb} = -\frac{qN_B\epsilon_{si}A^2}{Q - Q_{bi}}. \tag{30}$$

The collector–emitter redistribution capacitance C_{cer} is solved from the ambipolar diffusion equation as

$$C_{cer} = \frac{QC_{bcj}}{3Q_B} \tag{31}$$

where Q_B is the background mobile carrier base charge and C_{bcj} is the base–collector depletion capacitance. The carrier multiplication charge and capacitance relating to C_{cer} are given as

$$Q_{\rm mult} = (M-1)Q_{ce} \text{ and } C_{\rm mult} = (M-1)C_{cer}. \tag{32}$$

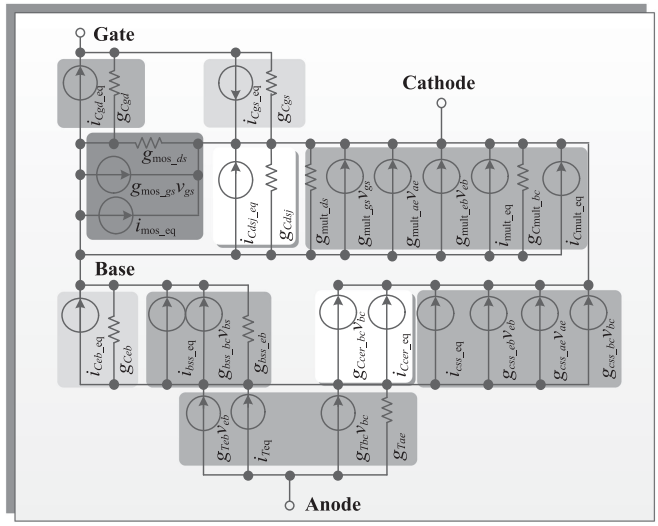


Fig. 5. Discretized and linearized equivalent circuit of IGBT (IGBT-DLE).

2) Model Discretization and Linearization: After applying the Newton–Raphson method on four current sources and the conductivity-modulated base resistance r_b, the analog equivalent circuit model (IGBT-AE) shown in Fig. 4(b), containing five nonlinear and time-varying elements, is transferred into discretized and linearized equivalent circuits (IGBT-DLE), as shown in Fig. 5. The iterative equations of $i_{\rm mos}$, i_T, i_{css}, i_{bss}, $i_{\rm mult}$ for the $(n+1)$th iteration are obtained as follows:

$$i_{\rm mos}^{n+1} = i_{mos\_eq}^{n} + g_{\rm mos\_gs}^{n} v_{gs}^{n+1} + g_{\rm mos\_ds}^{n} v_{ds}^{n+1} \tag{33}$$

$$i_T^{n+1} = i_{T\rm eq}^{n} + g_{Tae}^{n} v_{ae}^{n+1} + g_{Tbc}^{n} v_{bc}^{n+1} + g_{Teb}^{n} v_{eb}^{n+1} \tag{34}$$

$$i_{css}^{n+1} = i_{css\_eq}^{n} + g_{css\_bc}^{n} v_{bc}^{n+1} + g_{css\_ae}^{n} v_{ae}^{n+1} + g_{css\_eb}^{n} v_{eb}^{n+1} \tag{35}$$

$$i_{bss}^{n+1} = i_{bss\_eq}^{n} + i_{bss\_eb}^{n} v_{eb}^{n+1} + g_{bss\_bc}^{n} v_{bc}^{n+1} \tag{36}$$

$$i_{\rm mult}^{n+1} = i_{\rm mult\_eq}^{n} + g_{\rm mult\_ds}^{n} v_{ds}^{n+1} + g_{\rm mult\_ds}^{n} v_{ds}^{n+1} + g_{\rm mult\_ae}^{n} v_{ae}^{n+1} + g_{\rm mult\_eb}^{n} v_{eb}^{n+1}. \tag{37}$$

Applying KCL to the nodes gate, collector, base and emitter, results in the following nodal equation:

$$\boldsymbol{G}^{\rm IGBT} \cdot \boldsymbol{V}^{\rm IGBT} = \boldsymbol{I}_{eq}^{\rm IGBT} \tag{38}$$

where

$$\boldsymbol{V}^{\rm IGBT} = [v_c \quad v_g \quad v_a \quad v_d \quad v_e]^T \tag{39}$$

$\boldsymbol{I}_{eq}^{\rm IGBT}$ is given in (40) as shown at bottom of the next page and the 5×5 conductance matrix is given in (41) as shown at bottom of the next page.

Applying the Newton–Raphson method to solve the nonlinear matrix equation (38), $\Delta \boldsymbol{V}^{\rm IGBT}$ is obtained to update $\boldsymbol{V}^{\rm IGBT}$

$$Q_{gd} = \begin{cases} C_{oxd} v_{dg}, & v_{ds} \le v_{gs} - v_{Td} \\ \frac{qN_B\epsilon_{si}A_{gd}^2}{C_{oxd}} \left[\frac{C_{oxd}W_{gdj}}{\epsilon_{si}A_{gd}} - \ln\left(1 + \frac{C_{oxd}W_{gdj}}{\epsilon_{si}A_{gd}}\right)\right] - C_{oxd}v_{Td}, & v_{ds} > v_{gs} - v_{Td} \end{cases} \tag{27}$$

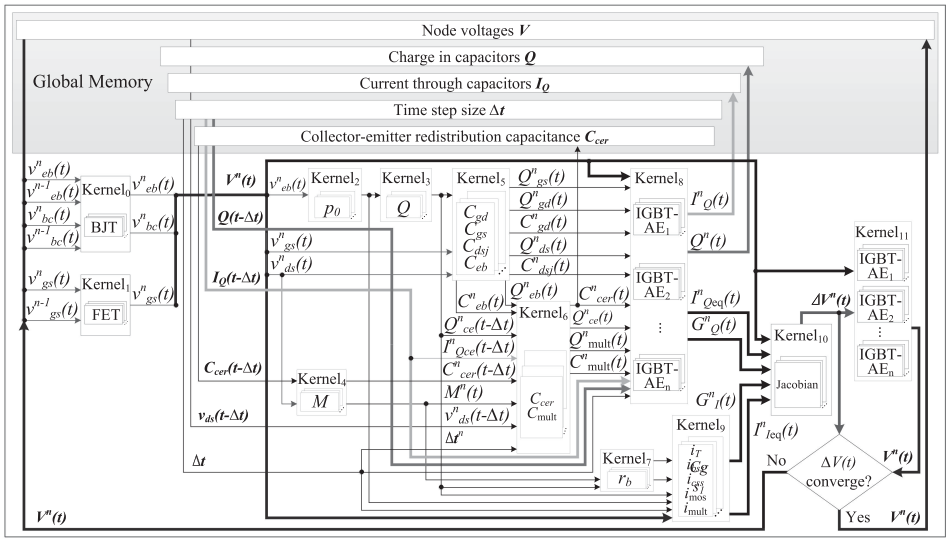


Fig. 6. Massive thread parallel implementation of the physics-based IGBT model.

iteratively. Therefore, the iterative equation is given as

$$\boldsymbol{G}^{\mathrm{IGBT(n)}}\Delta\boldsymbol{V}^{\mathrm{IGBT(n+1)}} = -\boldsymbol{I}^{\mathrm{IGBT(n)}} \tag{42}$$

where

$$\boldsymbol{I}^{\mathrm{IGBT(n)}} = \boldsymbol{G}^{\mathrm{IGBT(n)}}\boldsymbol{V}^{\mathrm{IGBT(n)}} - \boldsymbol{I}_{eq}^{\mathrm{IGBT}(n)}. \tag{43}$$

3) Parallel Massive-Thread Mapping: For a system containing *n* IGBTs, the massive-thread parallel implementation is shown in Fig. 6 and is described in Algorithm 2. Among the 12 kernels involved in the module, Kernel_0 and Kernel_1 check p-i-n junction and MOSFET junction voltage limitation within successive Newton–Raphson iterations; intermediate parameters p_0, Q, and M are processed in Kernel_2, Kernel_3 and Kernel_4; six nonlinear capacitors C_{gd}, C_{gs}, C_{dsj}, C_{mult}, C_{eb}, and C_{cer} are updated in Kernel_5 and Kernel_6; with those capacitances, the equivalent conductance $\boldsymbol{G}_Q$ and parallel current source $\boldsymbol{I}_{Q\mathrm{eq}}$

$$\boldsymbol{I}_{eq}^{\mathrm{IGBT}} = \begin{bmatrix} i_{c\_\mathrm{eq}} \\ i_{g\_\mathrm{eq}} \\ i_{a\_\mathrm{eq}} \\ i_{d\_\mathrm{eq}} \\ i_{e\_\mathrm{eq}} \end{bmatrix} = \begin{bmatrix} i_{\mathrm{mult\_eq}} + i_{C\mathrm{mult\_eq}} + i_{css\_\mathrm{eq}} + i_{C\_cer\_\mathrm{eq}} + i_{Cgs\_\mathrm{eq}} + i_{Cdsj\_\mathrm{eq}} + i_{\mathrm{mos\_eq}} \\ -i_{Cgs\_\mathrm{eq}} + i_{Cdg\_\mathrm{eq}} \\ -i_{T\mathrm{eq}} \\ i_{C\_ceb\_\mathrm{eq}} + i_{bss\_\mathrm{eq}} - i_{\mathrm{mos\_eq}} - i_{Cdg\_\mathrm{eq}} - i_{Cdsj\_\mathrm{eq}} - i_{\mathrm{mult\_eq}} - i_{C\mathrm{mult\_eq}} \\ i_{css\_\mathrm{eq}} - i_{bss\_\mathrm{eq}} - i_{C\_cer\_\mathrm{eq}} - i_{C\_ceb\_\mathrm{eq}} + i_{T\mathrm{eq}} \end{bmatrix} \tag{40}$$

$$\boldsymbol{G}^{\mathrm{IGBT}} = \begin{bmatrix} g_{\mathrm{mult\_ds}} + g_{\mathrm{mult\_gs}} + g_{\mathrm{mult\_bc}} + g_{css\_bc} + g_{C\_cer\_bc} + g_{Cgs} + g_{Cdsj} + g_{\mathrm{mos\_ds}} + g_{\mathrm{mos\_gs}} & -g_{\mathrm{mult\_gs}} - g_{Cgs} - g_{\mathrm{mos\_gs}} & -g_{\mathrm{mult\_ae}} - g_{css\_ae} & -g_{\mathrm{mult\_ds}} - g_{C\mathrm{mult}\_bc} - g_{css\_bc} + g_{css\_eb} - g_{C\_cer\_bc} - g_{dsj} - g_{\mathrm{mos\_ds}} + g_{\mathrm{mult\_eb}} & -g_{\mathrm{mult\_ae}} - g_{\mathrm{css}\_ae} + g_{\mathrm{css}\_eb} - g_{\mathrm{mult\_eb}} \\ -g_{Cgs} & g_{Cgs} + g_{Cdg} & 0 & -g_{Cdg} & 0 \\ -g_{Tbc} & 0 & g_{Tae} & g_{Tbc} - g_{Teb} & -g_{Tae} + g_{Teb} \\ g_{bss\_bc} - g_{\mathrm{mos\_gs}} - g_{\mathrm{mos\_ds}} - g_{C\_dsj} - g_{\mathrm{mult\_ds}} - g_{\mathrm{mult\_gs}} - g_{C\mathrm{mult}\_bc} & g_{\mathrm{mos\_gs}} - g_{Cdg} + g_{\mathrm{mult\_gs}} & g_{\mathrm{mult\_ae}} & g_{Ceb} - g_{bss\_bc} + g_{bss\_eb} + g_{\mathrm{mos\_ds}} + g_{Cdg} + g_{Cdsj} + g_{\mathrm{mult\_ds}} - g_{\mathrm{mult\_eb}} - g_{C\mathrm{mult}\_bc} & -g_{Ceb} + g_{bss\_eb} - g_{\mathrm{mult\_ae}} + g_{\mathrm{mult\_eb}} \\ -g_{css\_bc} - g_{C\_cer\_bc} - g_{bss\_bc} + g_{Tbc} & 0 & g_{css\_ae} - g_{Tae} & g_{css\_bc} - g_{css\_eb} + g_{C\_cer\_bc} - g_{bss\_eb} + g_{bss\_bc} - g_{Ceb} - g_{Tbc} + g_{\mathrm{Teb}} & -g_{\mathrm{css}\_ae} + g_{\mathrm{css}\_eb} + g_{bss\_eb} + g_{Ceb} + g_{Tae} - g_{\mathrm{Teb}} \end{bmatrix} \tag{41}$$

Algorithm 2: IGBT Kernel.

procedure Physics-based IGBT module
Check p-n junction voltage $v_{eb}(t)$ and $v_{bc}(t)$ from last iteration value ▷ Kernel_0
Check MOSFET junction voltage $v_{gs}(t)$ from last iteration value ▷ Kernel_1
Solve p_0 ▷ Kernel_2
Calculate Q, Q_1 and Q_2 ▷ Kernel_3
Update intermediate parameter M ▷ Kernel_4
Calculate $\boldsymbol{C}$ and $\boldsymbol{Q}$ in C_{gd}, C_{gs}, C_{dsj}, and C_{eb} ▷ Kernel_5
Calculate $\boldsymbol{C}$ and $\boldsymbol{Q}$ in C_{cer} and C_{mult} ▷ Kernel_6
Calculate r_b ▷ Kernel_7
Update $\boldsymbol{I}_Q$, $\boldsymbol{I}_{Q\text{eq}}$, and $\boldsymbol{G}_Q$ in all capacitors ▷ Kernel_8
Calculate $\boldsymbol{G}_I$ and $\boldsymbol{I}_{I\text{eq}}$ for all current sources ▷ Kernel_9
Build matrix equation (42) and solve for ΔV ▷ Kernel_{10}
Update $\boldsymbol{V}^{\text{IGBT}}(t)$ for current iteration ▷ Kernel_{11}
Check convergence of $\Delta \boldsymbol{V}^{\text{IGBT}}$
  if $\Delta \boldsymbol{V}^{\text{IGBT}}$ converges **then**
    Store $\boldsymbol{V}^{\text{IGBT}}$ to global memory and update t
  else
    Start from checking junction iteration limit

are updated in Kernel_8; Kernel_7 calculates the base resistance r_b for Kernel_9, where $\boldsymbol{G}_I$ and $\boldsymbol{I}_{I\text{eq}}$ are updated with the approximation to current sources i_{mos}, i_T, i_{css}, i_{bss}, and i_{mult}. Using all the resistances, charges, currents, equivalent conductances, and parallel current sources calculated above, Kernel_{10} builds the iterative Jacobian matrix equation in (42) and solves for $\Delta \boldsymbol{V}^{\text{IGBT}}$ using Gaussian elimination and backward substitution, and Kernel_{11} assigns n CUDA blocks containing five threads per block to update the voltage vector for each IGBT. When $\Delta \boldsymbol{V}^{\text{IGBT}}$ converges, the iterative solution for the current time step is accomplished and node voltages (NVs) $\boldsymbol{V}^{\text{IGBT}}$ are stored into global memory and used as the initial values for the next time step; otherwise, the iteration is repeated from checking junction limit as the initial value.

C. MMC and Control Strategy

1) Circuit Structure: Commonly utilized in HVdc systems, the MMC circuit consists of SMs, which contains two IGBT-diode units, as shown in Fig. 7. The MMC circuit has been modeled by different types of models that include equivalent circuit based model, switching function model, and averaged value model [30]–[33]. Due to the most detailed information inside every device and the nanosecond time step, the physics-based model requires so much computational burden that it is not frequently used in simulation. However, the application of GPU makes its application feasible in both accuracy and speed.

Fig. 7 shows a three-phase cascade MMC circuit structure consisting of n half-bridge submodules (SM) in each arm, with each SM containing two IGBTs, two antiparallel diodes, and an energy storage capacitor. Based on the gate signal combination and current direction of each IGBT, the SMs have different operating states. Once S_1 gate has ON signal and S_2 gate has OFF signal, C_m will be charged and discharged according to

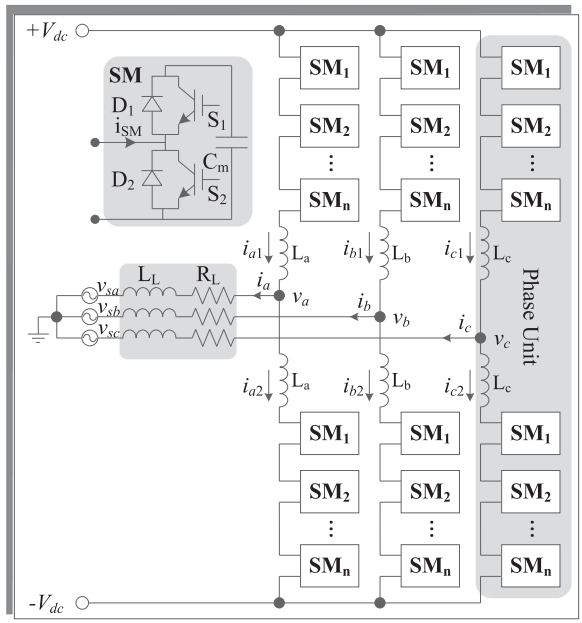


Fig. 7. MMC circuit structure.

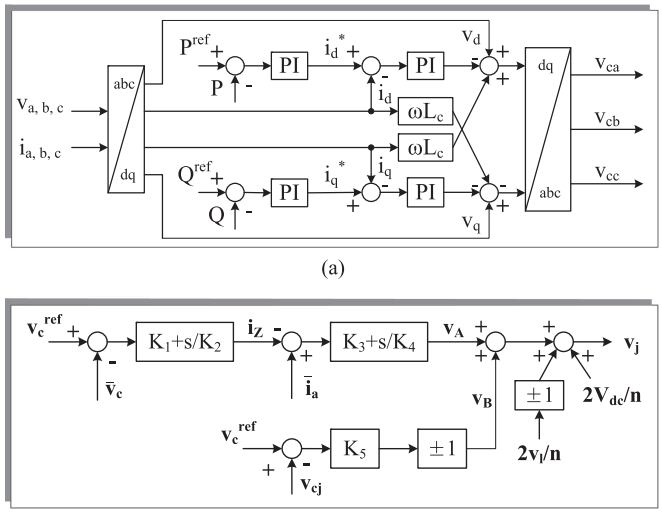


Fig. 8. MMC control scheme: (a) active and reactive power control of the MMC; (b) averaging and balancing control of the MMC.

the direction of i_{SM}; when S_1 gate has OFF signal and S_2 gate has ON signal, C_m will be discharged regardless of the current direction. Furthermore, when gate signal combination is "00," the SM is blocked and not used in normal operation; and the gate signal combination "11" causes a short circuit of the SM capacitor.

The control strategies of MMC circuit adopted in this paper include the active and reactive power control, capacitor voltage averaging, and balancing control [35]–[37]. The outer loop in Fig. 8(a) is the active power controller; and the inner loop is the current controller. Given a fixed power reference, the difference of active and reactive powers is input to proportional-integral controllers to produce reference *dq* currents. After producing

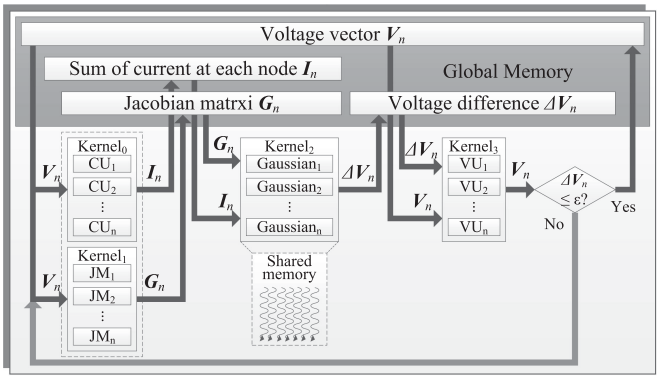


Fig. 9. Massive-thread parallel implementation of the Newton–Raphson method.

the *dq* voltages, the *abc* phase reference voltages are calculated for modulation. In the capacitor voltage averaging and balanced control, as shown in Fig. 8(b), the averaging control signal V_A ensures that the average voltage value of all the capacitors in each phase track the command value; and the balancing control signal V_B employs the phase-shifted carrier signals to force the dc voltage of each capacitor to follow the reference value.

D. Massive-Thread Parallel Implementation of the Newton–Raphson Method

1) Algorithm: To find the solution of a nonlinear system $\boldsymbol{F}(X)$ consisting of k unknown variables, utilizing the Jacobian matrix $\boldsymbol{J}_F(X)$ for linear approximation gives the following equation:

$$0 = \boldsymbol{F}(\boldsymbol{X}^n) + \boldsymbol{J}_F(\boldsymbol{X}^n)(\boldsymbol{X}^{n+1} - \boldsymbol{X}^n). \tag{44}$$

The Jacobian matrix $\boldsymbol{J}_F(\boldsymbol{X})$ is a $k \times k$ matrix of first-order partial derivatives of $\boldsymbol{F}$ given as

$$\boldsymbol{J}_F = \frac{d\boldsymbol{F}}{d\boldsymbol{X}} = \begin{bmatrix} \dfrac{\partial F_1}{\partial X_1} & \cdots & \dfrac{\partial F_1}{\partial X_k} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial F_k}{\partial X_1} & \cdots & \dfrac{\partial F_k}{\partial X_k} \end{bmatrix}. \tag{45}$$

Solving for the root of $\boldsymbol{F}(X)$ is numerically replaced by solving (44) for $(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)$ and updating $\boldsymbol{X}_{n+1}$ from $\boldsymbol{X}_n$ The solution process is repeated until the difference $||\boldsymbol{X}_{n+1} - \boldsymbol{X}_n||$ is converged.

According to KCL,

$$\Sigma \boldsymbol{I}(\boldsymbol{V}) = 0 \tag{46}$$

where $\boldsymbol{I}(\boldsymbol{V})$ refers to the sum of currents leaving each node. Applying (44) to (46) results in

$$\boldsymbol{J}_V{}^n(\boldsymbol{V}^{n+1} - \boldsymbol{V}^n) = -\Sigma \boldsymbol{I}^n. \tag{47}$$

In the nonlinear physics-based IGBT model, the voltage vector for the solution of KCL at each node of collector, gate, anode, drain, and emitter is given as

$$-\boldsymbol{V}^n = [v^{c(n)} \quad v^{g(n)} \quad v^{a(n)} \quad v^{d(n)} \quad v^{e(n)}]^T.$$

Applying the Newton–Raphson iterative equation results in

$$\boldsymbol{G}^{\mathrm{IGBT}(n)}(\boldsymbol{V}^{n+1} - \boldsymbol{V}^n) = -\boldsymbol{I}^n \tag{48}$$

where the conductance matrix $\boldsymbol{G}^{\mathrm{IGBT}(n)}$ is the Jacobian matrix $\boldsymbol{J}_V{}^n$, and the right-hand side vector is given as

$$-\boldsymbol{I}^n = [i^{c(n)} \quad i^{g(n)} \quad i^{a(n)} \quad i^{d(n)} \quad i^{e(n)}]^T.$$

See (49) and (50) shown at the bottom of this page.

$$\boldsymbol{G}_{SM}^{\mathrm{orig}} = \begin{bmatrix} G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & G_{1,5} & G_{1,6} & G_{1,7} & G_{1,8} & G_{1,9} & G_{1,10} & G_{1,11} \\ G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & G_{2,5} & G_{2,6} & & & & & \\ G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & G_{3,5} & G_{3,6} & & & & & \\ G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & G_{4,5} & G_{4,6} & & & & & \\ G_{5,1} & G_{5,2} & G_{5,3} & G_{5,4} & G_{5,5} & G_{5,6} & & & & & \\ G_{6,1} & G_{6,2} & G_{6,3} & G_{6,4} & G_{6,5} & G_{6,6} & & & & & G_{6,11} \\ G_{7,1} & & & & & & G_{7,7} & G_{7,8} & G_{7,9} & G_{7,10} & G_{7,11} \\ G_{8,1} & & & & & & G_{8,7} & G_{8,8} & G_{8,9} & G_{8,10} & G_{8,11} \\ G_{9,1} & & & & & & G_{9,7} & G_{9,8} & G_{9,9} & G_{9,10} & G_{9,11} \\ G_{10,1} & & & & & & G_{10,7} & G_{10,8} & G_{10,9} & G_{10,10} & G_{10,11} \\ G_{11,1} & & & & & G_{11,6} & G_{11,7} & G_{11,8} & G_{11,9} & G_{11,10} & G_{11,11} \end{bmatrix} \tag{49}$$

$$\boldsymbol{G}_{SM} = \begin{bmatrix} G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & G_{1,5} & G_{1,6} & G_{1,7} & G_{1,8} & G_{1,9} & G_{1,10} & G_{1,11} \\ G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & G_{2,5} & G_{2,6} & & & & & \\ G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & G_{3,5} & G_{3,6} & & & & & \\ G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & G_{4,5} & G_{4,6} & & & & & \\ G_{5,1} & G_{5,2} & G_{5,3} & G_{5,4} & G_{5,5} & G_{5,6} & & & & & \\ G_{6,1} & G_{6,2} & G_{6,3} & G_{6,4} & G_{6,5} & G_{6,6} & & & & & \\ G_{7,1} & & & & & & G_{7,7} & G_{7,8} & G_{7,9} & G_{7,10} & G_{7,11} \\ G_{8,1} & & & & & & G_{8,7} & G_{8,8} & G_{8,9} & G_{8,10} & G_{8,11} \\ G_{9,1} & & & & & & G_{9,7} & G_{9,8} & G_{9,9} & G_{9,10} & G_{9,11} \\ G_{10,1} & & & & & & G_{10,7} & G_{10,8} & G_{10,9} & G_{10,10} & G_{10,11} \\ G_{11,1} & & & & & & G_{11,7} & G_{11,8} & G_{11,9} & G_{11,10} & G_{11,11} \end{bmatrix} \tag{50}$$

Algorithm 3: Newton-Raphson Kernel.

procedure N-R ITERATION
`Iterative loop:`
Calculate $\boldsymbol{F}(\boldsymbol{X}^n)$ ▷ Kernel$_0$
Calculate $\boldsymbol{J}_F(\boldsymbol{X}^n)$ from $\boldsymbol{X}^n$ ▷ Kernel$_1$
Copy $\boldsymbol{J}_F(\boldsymbol{X}^n)$ and $\boldsymbol{F}(\boldsymbol{X}^n)$ into shared memory
Solve $\Delta\boldsymbol{X}^n$ in $\boldsymbol{J}_F(\boldsymbol{X}^n)\Delta\boldsymbol{X}^n = -\boldsymbol{F}(\boldsymbol{X}^n)$ ▷ Kernel$_2$
Update $\boldsymbol{X}^{n+1} \leftarrow \boldsymbol{X}^n + \Delta\boldsymbol{X}^n$
Calculate $||\Delta\boldsymbol{X}^n||$ ▷ Kernel$_3$
Store $||\Delta\boldsymbol{X}^n||$ into global memory
if $||\Delta\boldsymbol{X}^n|| < \epsilon$ **then**
$t \leftarrow t + \Delta t$
else
go to `Iterative loop`

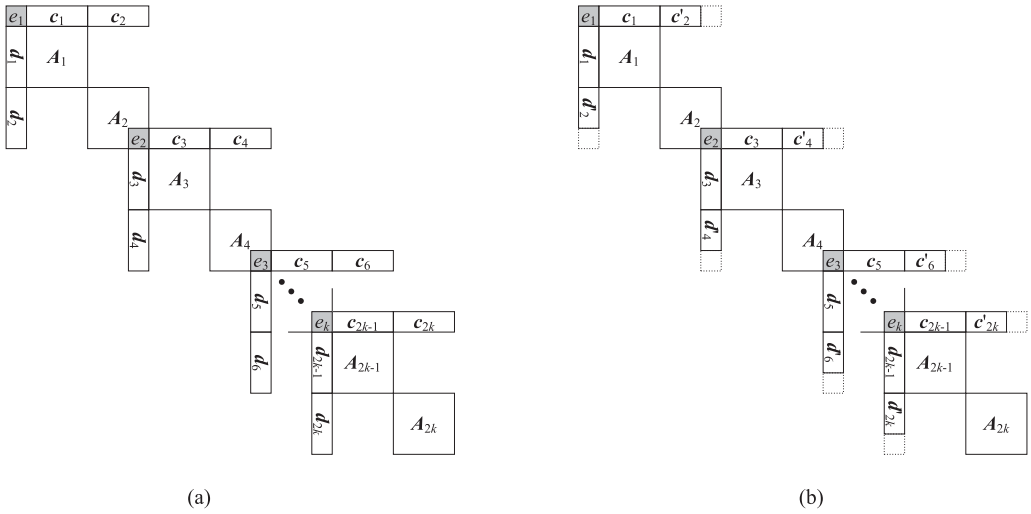


Fig. 10. Jacobian matrices for the MMC circuit: (a) Original $\boldsymbol{G}_{\text{MMC}}$; (b) updated $\boldsymbol{G}^*_{\text{MMC}}$ using relaxation.

2) Massive-Thread Parallel Implementation: In the massive-thread parallel module for the Newton–Raphson method, four kernels are involved, as shown in Fig. 9 and Algorithm 3. First, Kernel $_0$ and Kernel$_1$ update $\boldsymbol{I}^n$, which equals the sum of currents leaving the nodes, and the Jacobian matrix $\boldsymbol{J}_F(\boldsymbol{X}^n)$, which is the conductance matrix with current units (CUs) and Jacobian matrix units (JMs), respectively. All parameters are copied to shared memory for Kernel$_2$ to solve (44) by Gaussian elimination method. Then, $\boldsymbol{V}^{n+1}$ is updated and verified if it satisfies the convergence criteria in Kernel$_3$ with voltage units (VUs). The iterations continue until $\boldsymbol{V}$ is converged or the maximum number of iterations is reached.

E. Block Jacobian Matrix Decomposition for MMC

Applying the physics-based IGBT and power diode module mentioned above, each SM has five nodes for the IGBT and one more inside node for the power diode, which results in the Jacobian matrix (the original G matrix for the SM) (49).

1) Matrix Update Using a Relaxation Algorithm: Although the sparse matrix G^{orig}_{SM} is irregular, it can be reshaped by eliminating two elements, $G_{6,11}$ and $G_{11,6}$ brought about by the capacitance, using the relaxation algorithm. After the transformation, the linearized equation of an SM is given as

$$\boldsymbol{G}^{\text{orig(n)}}_{SM} \cdot \Delta\boldsymbol{V}^{n+1} = -\boldsymbol{I}^{\text{orig(n)}} \tag{51}$$

is updated to

$$\boldsymbol{G}^{n}_{SM} \cdot \Delta\boldsymbol{V}^{n+1} = -\boldsymbol{I}^{n} \tag{52}$$

where $\boldsymbol{G}_{SM}$ is given in (50) and $\boldsymbol{I}^n$ comes from $\boldsymbol{I}^{\text{orig(n)}}$ whose 6th and 11th elements are adjusted by $G_6\Delta v^n_6$ and $G_{11}\Delta v^n_{11}$ with known values from previous iteration. Although there is an outer Gauss–Jacobi loop over the Newton–Raphson iterations to guarantee the convergence of Δv^n, the updated $\boldsymbol{G}_{SM}$ results in a better bordered-block pattern, which benefits to the parallel implementation on the GPU.

2) Partial LU Decomposition for MMC: For a k-SM MMC circuit, each cascade node connects to two nonlinear IGBT-diode units. Therefore, the complete structure of the Jacobian matrix $\boldsymbol{G}_{\text{MMC}}$ is shown in Fig. 10(a). The connections on Node 1 and Node 11 of every G_{SM} are relaxed to decouple the large Jacobian matrix. Thus, the updated Jacobian matrix $\boldsymbol{G}^*_{\text{MMC}}$ is shown in Fig. 10(b), and the Newton–Raphson linearized equation, given as

$$\boldsymbol{G}^{n}_{\text{MMC}} \cdot \Delta\boldsymbol{V}^{n+1} = -\boldsymbol{I}^{n} \tag{53}$$

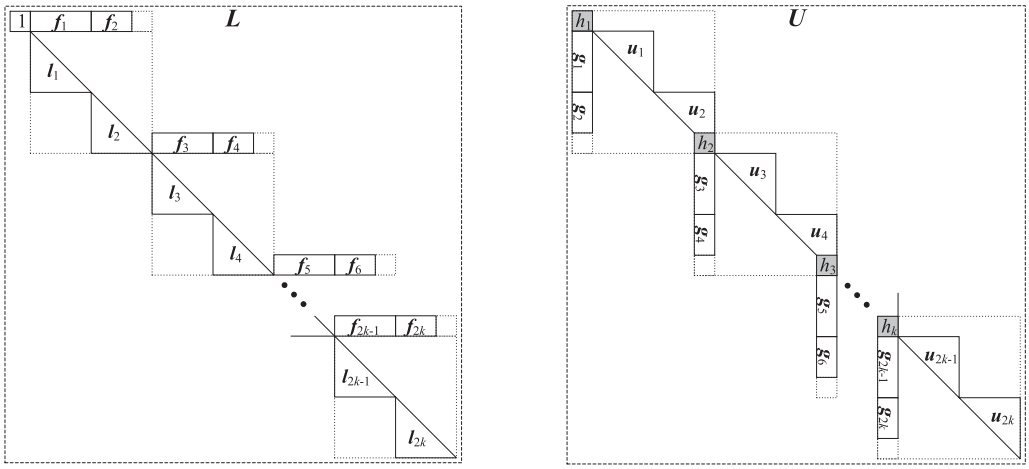
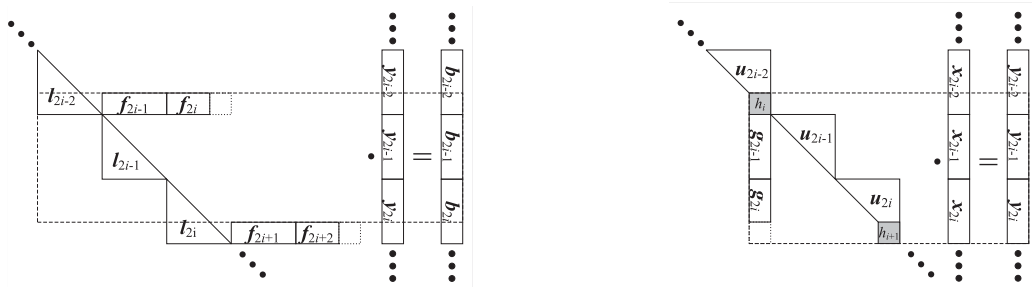


Fig. 11. Partial LU decomposition for $\boldsymbol{G}^{*}_{\mathrm{MMC}}$.



(a) (b)

Fig. 12. Blocked linear MMC system solver: (a) Blocked forward substitution; (b) blocked backward substitution.

is also updated as

$$\boldsymbol{G}^{*(n)}_{\mathrm{MMC}} \cdot \Delta \boldsymbol{V}^{n+1} = -\boldsymbol{I}^{*(n)} \tag{54}$$

where $-\boldsymbol{I}^{*(n)}$ is $-\boldsymbol{I}^{n}$ adjusted by previous iteration values. Processing the LU decompositions for all A matrices in $\boldsymbol{G}^{*}_{\mathrm{MMC}}$, we obtain

$$\boldsymbol{l}_j \cdot \boldsymbol{u}_j = \boldsymbol{A}_j \quad (j = 1, 2, \cdots, 2k). \tag{55}$$

Then, the border vectors, $\boldsymbol{f}_j$ and $\boldsymbol{g}_j$ in Fig. 11, can be found according to the following relations:

$$\begin{cases} \boldsymbol{f}_j \cdot \boldsymbol{u}_j = \boldsymbol{c}_j \\ \boldsymbol{l}_j \cdot \boldsymbol{g}_j = \boldsymbol{d}_j \end{cases}. \tag{56}$$

Since $\boldsymbol{l}_j$ and $\boldsymbol{u}_j$ are lower and upper triangular matrices, $\boldsymbol{f}_j$ and $\boldsymbol{g}_j$ can be solved with forward and backward substitutions. Finally, the elements at the connecting nodes, h_i $(i = 1, 2, \ldots, k)$ in Fig. 11, are calculated with e_i in Fig. 10 as

$$h_i = e_i - \boldsymbol{f}_{2i-1} \cdot \boldsymbol{g}_{2i-1} - \boldsymbol{f}_{2i} \cdot \boldsymbol{g}_{2i}. \tag{57}$$

The proposed partial LU decomposition can be computed in parallel due to the decoupled structure of updated Jacobian matrix $\boldsymbol{G}^{*}_{\mathrm{MMC}}$, including the LU decomposition of $\boldsymbol{A}_j$, calculation of border vectors $\boldsymbol{f}_j$, $\boldsymbol{g}_j$, and updating of connecting node elements h_i.

3) Blocked Forward and Backward Substitutions: After obtaining the semilower and upper triangular matrices utilizing partial LU decomposition, blocked forward and backward substitutions are utilized to obtain the solution of the linear system. For the equation $\boldsymbol{LU} \cdot \boldsymbol{x} = \boldsymbol{b}$, defining

$$\boldsymbol{U} \cdot \boldsymbol{x} = \boldsymbol{y} \tag{58}$$

we obtain

$$\boldsymbol{U} \cdot \boldsymbol{y} = \boldsymbol{b}. \tag{59}$$

The blocked forward substitution shown in Fig. 12(a) is used to solve for $\boldsymbol{y}$ in (59). All elements of $\boldsymbol{y}_{2i-1}$ and most elements of $\boldsymbol{y}_{2i}$ except for the last one can be solved directly; then, the last element of previous block $\boldsymbol{y}_{2i-2}$ can be obtained with solved elements in $\boldsymbol{y}_{2i-1}$ and $\boldsymbol{y}_{2i}$; and so forth, the missing element of $\boldsymbol{y}_{2i}$ can be obtained from the solution of the next block. Fortunately, $\boldsymbol{y}_{2k}$ in the last block can be fully solved since it has no overlap border vector $\boldsymbol{f}$. Because of the decoupled structure of $\boldsymbol{L}$, the blocked forward substitution can be accomplished by GPU-based parallelism. Similarly, the solution of (58) can be obtained by the blocked backward substitution shown in Fig. 12(b) with $\boldsymbol{y}_j$. First, all connecting elements h_i can be solved in parallel. Second, the border vector $\boldsymbol{g}_j$ can be eliminated by updating the right-hand side vector. Finally, backward substitution is implemented on all blocked $\boldsymbol{U}_j$ in parallel.

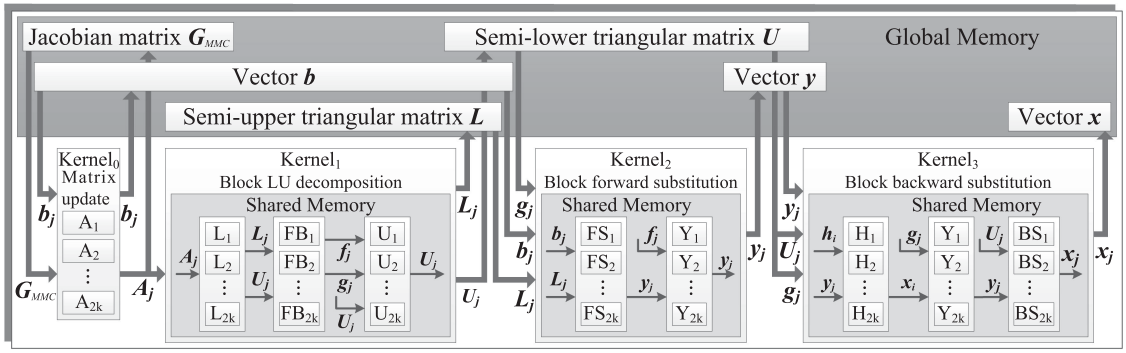


Fig. 13. Massive thread parallel implementation of partial LU decomposition.

Algorithm 4: Partial LU decomposition for MMC.

procedure SOLVE MATRIX EQUATION FOR MMC CIRCUIT USING PARTIAL LU DECOMPOSITION

`Produce semi-upper and semi-lower triangular matrices:`

Update matrix equation using relaxation algorithm ▷ Kernel$_0$

Block LU decomposition to get $\boldsymbol{L}_j$, $\boldsymbol{U}_j$
Backward and forward substitution for $\boldsymbol{f}_j$, $\boldsymbol{g}_j$
Update $\boldsymbol{U}_j$ with e_i ▷ Kernel$_1$

`Blocked forward substitution:`

Forward substitution in block
$\boldsymbol{y}_j$ update ▷ Kernel$_2$

`Blocked backward substitution:`

Compute h_i directly
Subtract effect of h_i to update $\boldsymbol{y}_j$
Backward substitution in block ▷ Kernel$_3$

In the MMC circuit, the size of the Jacobian matrix grows with the number of output voltage levels. Instead of solving a system containing a $(10k+1)\times(10k+1)$ large Jacobian matrix, the 5×5 block perfectly accommodates the parallel scheme of GPU with its limited shared memory to reduce the data transmission cost.

F. Parallel Massive-Thread Mapping

As shown in Fig. 13 and Algorithm 4, four kernels are involved in the parallel module for the partial LU decomposition method and blocked linear solution in MMC. Kernel$_0$ updates the matrix equation using the relaxation algorithm, and the original G_{MMC} matrix is reshaped to G^*_{MMC}, which is output to Kernel$_1$. To calculate the semi-lower and -upper triangular matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ in an MMC circuit containing k SMs, there are three steps inside Kernel$_1$ listed as follows:

1) normal LU decomposition in each block to get $\boldsymbol{L}_j$, where $j=(1,2,\cdots,2k)$;
2) backward substitution to obtain $\boldsymbol{f}_j$ from $\boldsymbol{U}_j$ and $\boldsymbol{c}_j$, forward substitution to obtain $\boldsymbol{g}_j$ from $\boldsymbol{L}_j$ and $\boldsymbol{d}_j$ in each block;
3) update $\boldsymbol{U}_j$ with e_i, where $i=(1,2,\ldots,k)$.

Kernel$_2$ has two steps as follows:

1) forward substitution in block for $\boldsymbol{y}_j$;
2) update $\boldsymbol{y}_j$ with border vector $\boldsymbol{f}_j$.

And Kernel$_3$ has the following three steps:

1) calculate h_i;
2) update $\boldsymbol{y}_j$ with h_i;
3) backward substitution in block

to obtain the final result $\boldsymbol{x}_j$.

G. Predictor–Corrector Variable Time-Stepping Scheme

A variable time-stepping scheme is used in this paper for numerical solution of the transient nonlinear system response to ensure both efficiency and accuracy. In most power electronic systems, the transient portion is much smaller than the steady-state part; thus, a dynamically adaptive time step reduces the calculation during steady state while maintaining accuracy during the transient stage.

To find the numerical solution of ordinary differential equations, the predictor–corrector method utilizes a polynomial prediction of the derivative from the previous solution to the current solution, followed by a refinement of the predicted value. The predictor step normally applies an explicit integration method, such as the forward Euler method, while the corrector step applies an implicit integration method, such as the backward Euler or the trapezoidal method. If the local truncation error (LTE) is out of the prespecified tolerance range, which means that the current time-step size is too large to obtain an accurate result, a refined time-step size is adopted to recalculate the current time solution.

The overall flowchart of the iterative process using the predictor–corrector variable time-stepping method (PCVTSM) is shown in Fig. 14. Based on the Gear's method [38], forward

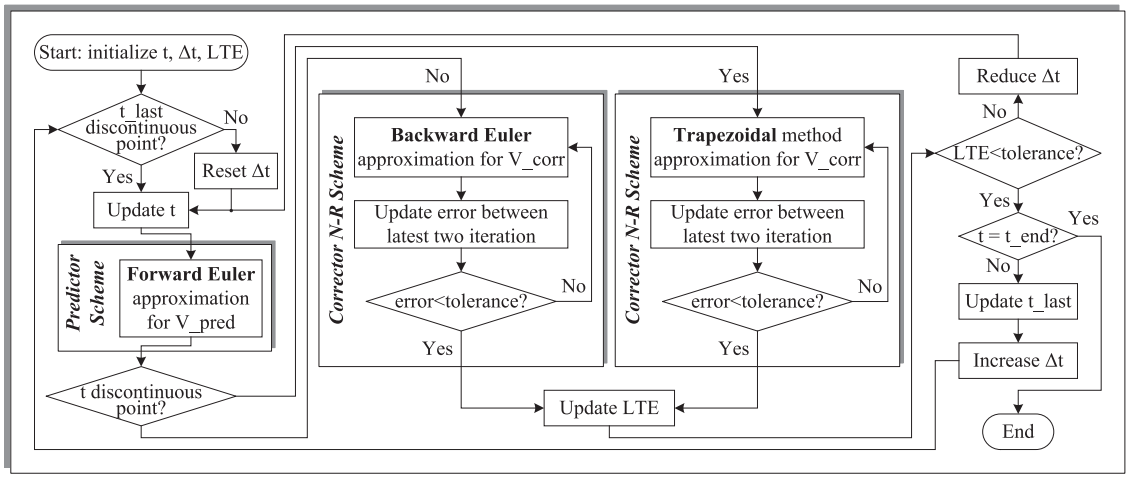


Fig. 14. Flowchart of variable time-stepping scheme.

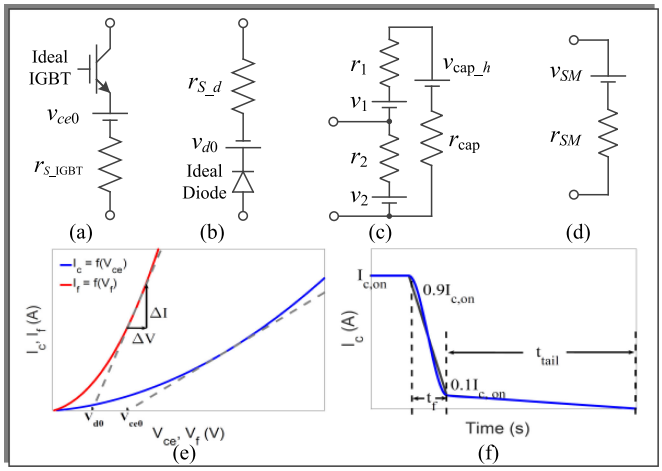


Fig. 15. (a) Behavior-based IGBT model. (b) Behavior-based power diode model. (c) Discretized SM circuit. (d) Equivalent TC of SM. (e) Linear approximation of IGBT and diode. (f) IGBT measurement model for turn-off current.

Euler is used to obtain the approximation from the values of previous time solution, t_last, as a predictor, V_pred. Then, two implicit methods backward Euler and trapezoidal method are utilized, depending on whether the current calculated solution is discontinuous or not, to obtain a corrector V_corr. If the LTE is reduced within the acceptable tolerance, the predictor–corrector routine is stopped and the end time t_end for the next time step is found to compute the numerical solution.

H. Behavior-Based IGBT Model and MMC Solution

1) Equivalent Model for SM: In many applications for fast simulation, the behavior-based IGBT and diode models are commonly adopted [39]. In this paper, such models are also demonstrated for system-level MMC circuit simulation. The IGBTs and diodes are represented as ideal switch models in series with voltage sources and resistors, as shown in Fig. 15(a) and (b). Based on the relationship between output voltages and currents from the IGBT and diode, a linear approximation of IGBT output voltage v_{ce} and current i_c, and diode output voltage v_F and current i_F can be obtained as follows:

$$r_{S\_\mathrm{IGBT}} = \frac{\Delta v_{ce}}{\Delta i_c} \tag{60}$$

$$r_{S\_d} = \frac{\Delta v_f}{\Delta i_f}. \tag{61}$$

The turn-off current is modeled using two linear slopes, as shown in Fig. 15(e). The capacitor in each SM is discretized into a resistor r_{cap} in series with a history voltage source $v_{cap\_h}(t-\Delta t)$. The $(r_1,\ v_1)$ and $(r_2,\ v_2)$ in Fig. 15(c) are decided by the gate signal and arm current direction. In this way, each SM's Thévenin equivalent circuit in Fig. 15(d) contains a time-dependent resistor r_{SM} and a history voltage source v_{SM} given as

$$r_{SM} = \frac{r_2(r_1 + r_{cap})}{r_1 + r_2 + r_{\mathrm{cap}}} \tag{62}$$

$$v_{SM} = \left(\frac{v_2}{r_2} + \frac{v_1 + v_{\mathrm{cap\_h}}(t-\Delta t)}{r_1 + r_{\mathrm{cap}}}\right) r_{SM} \tag{63}$$

where the capacitor history voltage can be updated as

$$v_{\mathrm{cap\_h}}(t-\Delta t) = v_{\mathrm{cap\_h}}(t-2\Delta t) + 2r_{\mathrm{cap}} i_{\mathrm{cap}}(t-\Delta t) \tag{64}$$

$$r_{\mathrm{cap}} = \frac{2\Delta t}{C}. \tag{65}$$

Thus, each arm of MMC containing n SMs in Fig. 7 is represented by a voltage source and a resistor as

$$v_{\mathrm{arm}} = \sum_{i=1}^{n} v_{SMi} \tag{66}$$

$$r_{\mathrm{arm}} = \sum_{i=1}^{n} r_{SMi}. \tag{67}$$

The arm current can be calculated with the above equivalent model. Since each SM's input current is the same as the arm current, the NV inside each SM can be updated in parallel using the solved arm current.

2) Parallel Massive-Thread Mapping: As shown in Fig. 16 and Algorithm 5, one kernel containing n threads is involved

Algorithm 5: Behavior-based MMC solution kernel.

while $t < t_{end}$ **do**
Update v_1, r_1 and v_2, r_2 in each SM
Calculate the $v_{\rm SM}(t)$, $r_{\rm SM}(t)$ in each SM
Sum up n SMs in each arm to solve $i_{\rm arm}(t)$ ▷ Kernel$_0$
Calculate node voltages for each SM
$t \leftarrow t + \Delta t$

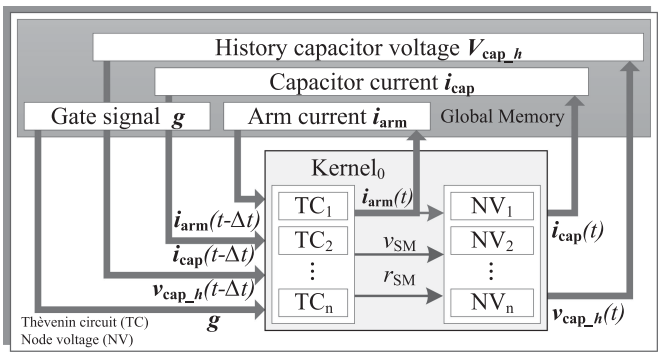


Fig. 16. Massive thread parallel implementation of behavior-based MMC circuit.

to solve the MMC circuit with behavior-based models. For a system with n SMs, (v_1, r_1) and (v_2, r_2) in each SM are decided according to gate signals and arm current directions, from which the equivalent Thévenin circuits (TC) are obtained using (62) and (63). From the lumped voltages and resistances of n SMs, the arm currents are solved for updating the NVs of each SM. Since there are only two node voltages that need to be calculated, one execution thread is required for each SM. Therefore, n threads are applied in the CUDA Kernel.

IV. Case Studies and Data Analysis

A. Test Cases for Physics-Based and Behavior-Based MMC Circuit Simulation

With the physics-based model, the size of the Jacobian matrix in the Newton–Raphson method is enlarged with the increasing number of SMs in the MMC circuit. For a single-phase l-level MMC system, there are $2(l-1)$ half-bridge SMs used, for which the size of Jacobian matrix $\boldsymbol{G}_{\rm MMC}$ is $(20l-19)\times(20l-19)$.

Due to the variety and difference of the conductance caused by the switching nature of IGBT, $\boldsymbol{G}_{\rm MMC}$ is very ill conditioned, whose condition number is normally more than 10^6. Therefore, the solution of (53) is very sensitive to errors and the convergence of Newton iterations is difficult; this situation becomes worse as the number of MMC levels is increased. SaberRD could not give complete simulation results when the MMC circuit became larger than single-phase five-level, or eight half-bridge SMs with the physics-based model, even when the parameters of LTE and the target iteration number in variable time-stepping method are properly adjusted, which could alleviate some convergence problems. Although the solution sensitivity convergence problem still exist in the proposed solution method with relaxation-based partial LU decomposition, the computable scale of MMC circuit is extended up to 3-phase 11-level, because solving the linearized Newton–Raphson equation block-by-block in small sizes reduces the probability of divergence significantly, instead of solving a large $(20l-19)\times(20l-19)$ linear system directly. Therefore, the 3-phase 11-level MMC circuit is the maximum scale that could be simulated on the GPU using the physics-based models. Since the largest system that could be successfully simulated in SaberRD is the single-phase five-level MMC circuit, the results and execution time comparison between the GPU code and SaberRD is based on that circuit. The MMC system parameters, including physics-based IGBT and power diode models are given in the appendix in Tables VII and VIII.

Due to the convergence limitation of the physics-based model, test cases larger than 3-phase 11-level MMC are modeled by the behavior-based model, and mutiple MMC circuits up to 1000 SMs per phase are tested to compare the performance between GPU and CPU codes for large-scale power electronic converter applications. The specifications and parameters of the behavior-based MMC circuit are listed in Table IX. A fixed time-step (10 μs) scheme is adopted for behavior-based model cases since the simulation time step (1 ns) is 10 000 times larger than the initial time step used for the physics-based model case.

B. Results and Comparison

The simulation results of single-phase five-level physics-based MMC circuit are shown in Fig. 17, including waveforms and analysis comparison between GPU and SaberRD. In Fig. 17(a) and (d), the waveforms of output voltage and load current show full agreement between the GPU (top panels) and SaberRD (bottom panels) results; zoomed-in waveforms are shown in Fig. 17(b) and (e). In order to analyze the steady-state harmonics by fast Fourier transform (FFT), the output voltages are resampled at 20 kHz, which is far higher than the 2.5 kHz PWM carrier signal f_c satisfying Nyquist's law, to produce the fixed sample rate signals, since the simulation output waveforms were obtained using the variable time-stepping method.

In the frequency spectrum diagram, as shown in Fig. 17(c) and (f), two major harmonic frequencies are the signals at 60 Hz and 2.5 kHz, which are the base system frequency and carrier signal frequency, respectively, and close agreement of magnitude of these two signals can be observed between the results obtained by GPU and SaberRD, respectively. The capacitor voltages are compared in Fig. 17(g) between GPU and SaberRD. With averaging and balancing control, the capacitor voltages are kept around the level of 2 $\rm V_{dc}/n_{arm}$, where $\rm V_{dc}$ is the dc voltage and $n_{\rm arm}$ is the number of SMs per arm. The device-level simulation results, IGBT gate OFF and ON waveforms, are given in Fig. 17(h) and (i), which show the detailed turn-on and turn-off voltage, current, and switching times. Table II lists the device

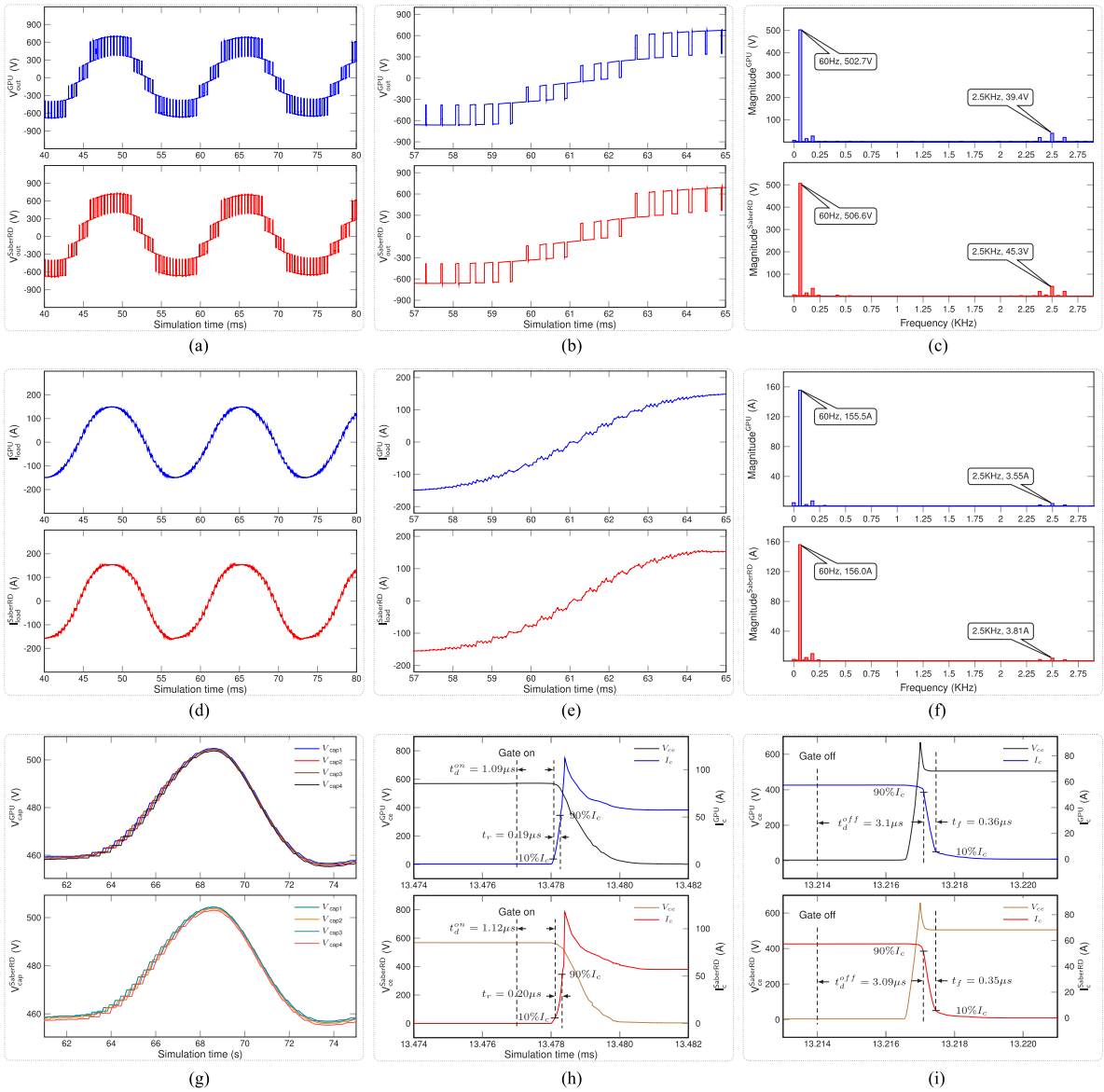


Fig. 17. Single-phase five-level physics-based MMC simulation results comparison between GPU (top) and SaberRD (bottom). (a) Five-level MMC V_{out} comparison. (b) Five-level MMC zoomed V_{out} comparison. (c) Five-level MMC V_{out} FFT comparison. (d) Five-level MMC I_{load} comparison. (e) Five-level MMC zoomed I_{load} comparison. (f) Five-level MMC I_{load} FFT comparison. (g) Five-level MMC arm V_{cap} comparison. (h) IGBT gate ON waveform comparison. (i) IGBT gate OFF waveform comparison.

TABLE II
DEVICE-LEVEL SWITCHING TIME AND POWER DISSIPATION FOR IGBT-DIODE UNIT

Switching time (μs)			Power dissipation (W)		
	SaberRD	GPU		SaberRD	GPU
$t_{d(\text{on})}^{\text{IGBT}}$	1.12	1.09	$P_{\text{on}}^{\text{IGBT}}$	112.31	113.02
t_{r}^{IGBT}	0.20	0.19	$P_{\text{off}}^{\text{IGBT}}$	74.01	74.84
$t_{d(\text{off})}^{\text{IGBT}}$	3.09	3.10	P_{cond}^{IGBT}	287.52	289.06
t_{f}^{IGBT}	0.35	0.36	P_{cond}^{Diode}	7.48	7.45
t_{rr}^{Diode}	0.66	0.64	P_{rr}^{Diode}	9.95	10.07

switching times and power dissipation of the lower IGBT-Diode unit in the SM, referred to as S_2 and D_2, respectively, in Fig. 7, during switching and conducting states, which are calculated using

$$P^{\text{IGBT}} = \frac{\int v_{ce}(t) i_c(t) dt}{T_s} \tag{68}$$

$$P^{\text{Diode}} = \frac{\int v_f(t) i_f(t) dt}{T_s} \tag{69}$$

for IGBT and diode, respectively, where T_s is the switching period.

The simulation results of 3-phase 11-level MMC circuit with nonlinear physics-based models, including output voltages and load currents, are shown in Fig. 18(a), which are compared with the simulation results of the same MMC circuit with behavior-based models shown in Fig. 18(b). The waveforms of physics-based model have much more detail than those from behavior-based model because of the nonlinear capacitance and dynamic current sources in the former model. When the MMC

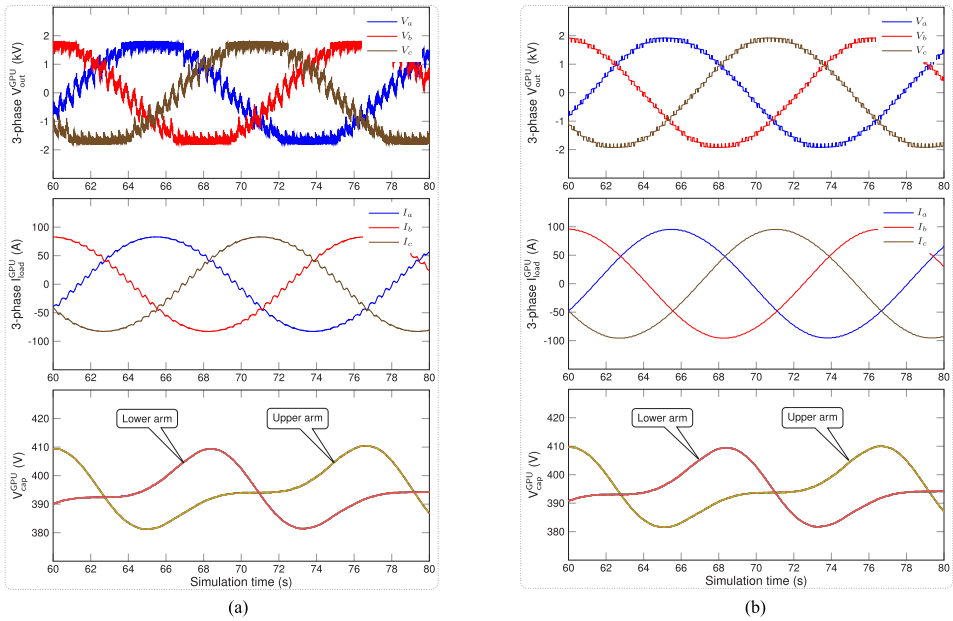


Fig. 18. 3-phase 11-level physics-based and behavior-based MMC simulation comparison. (a) 11-level physics-based MMC simulation. (b) 11-level behavior-based MMC simulation.

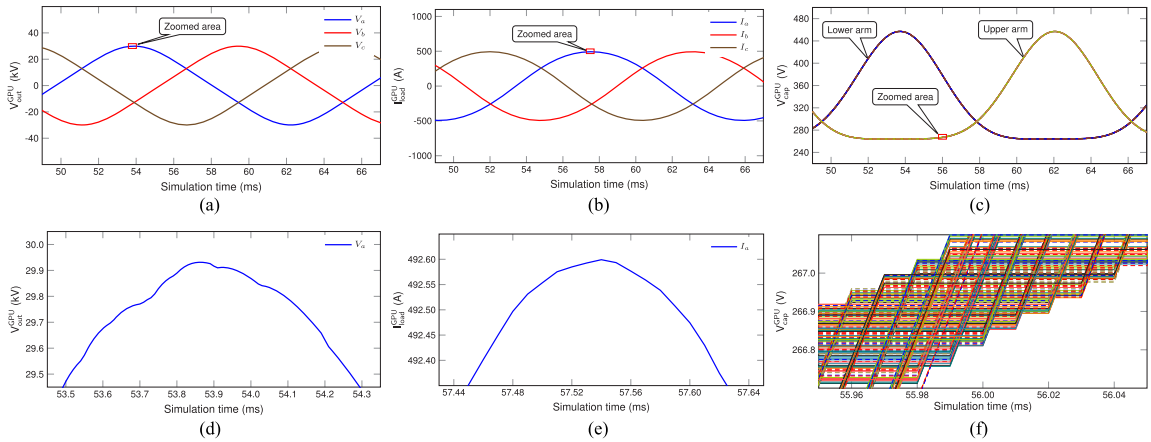


Fig. 19. 201-level behavior-based MMC output voltages $V_{\rm out}$, load currents $\rm I_{load}$, and arm capacitor voltages $V_{\rm cap}$. (a) 201-level behavior-based MMC $V_{\rm out}$. (b) 201-level behavior-based MMC $I_{\rm load}$. (c) 201-level behavior-based MMC arm $V_{\rm cap}$. (d) 201-level behavior-based MMC zoomed $V_{\rm out}$. (e) 201-level behavior-based MMC zoomed $I_{\rm load}$. (f) 201-level behavior-based MMC arm zoomed $V_{\rm cap}$.

levels increase beyond 11, the physics-based models are too sensitive to converge on not only SaberRD but also the proposed GPU simulator, which implies that the system Jacobian matrix of the Newton method is close to being singular.

The behavior-based 3-phase MMC circuits are tested from 11-level to 501-level. With the growth of level number, the output voltage and load current waveforms are highly close to being sinusoidal. Fig. 19(c) gives the simulation waveforms of capacitor voltages in the 201-level MMC circuit with behavior-based models, including lower and upper arms, and the waveforms inside the rectangular area are zoomed in Fig. 19(f) showing the detail of capacitor voltage change in the same arm.

C. Execution Time and Speedup Comparison

Table III lists the main performance metrics of the CPU and GPU processors used in this study. The single-phase MMC circuits are simulated from 2-level to 11-level as shown in Table IV for 100 ms using the variable time-stepping method, where the execution time of SaberRD, CPU (Intel i7-3770), Kepler GPU (Nvidia GTX Titan), and Pascal GPU (Nvidia GTX 1080) are listed under the headings SaberRD, CPU, $\rm GPU_K$, and $\rm GPU_P$, respectively. Since the simulation in SaberRD is hard to converge when the circuit contains more than eight SMs and

TABLE III
COMPARISON OF CPU AND GPU SPECIFICATIONS USED IN THIS PAPER

Processors	Base Clock (Hz)	Memory Data Rate (MT/s)	Computing Capability (GFLOPS)
i7-3770 (Ivy Bridge)	3.4 G	1600	108.8
GK110 (Kepler)	889 M	6008	5121
GP104 (Pascal)	1607 M	10 000	8228

TABLE IV
SABERRD, CPU, AND GPU EXECUTION TIMES FOR THE ONE-PHASE NONLINEAR PHYSICS-BASED MMC

N_L	N_{SM}	Execution time (s)				Speedup	
		SaberRD	CPU	GPU$_K$	GPU$_P$	GPU$_K$	GPU$_P$
2	2	53	50.8	143.1	70.9	0.35	0.72
3	4	100	99.9	162.4	81.6	0.62	1.22
4	6	149	143.8	175.5	87.4	0.82	1.65
5	8	202	192.5	184.7	90.2	1.04	2.13
6	10	-	240.9	195.2	94.1	1.23	2.56
7	12	-	297.3	212.7	101.3	1.40	2.93
8	14	-	351.8	227.5	107.5	1.55	3.27
9	16	-	411.5	237.9	111.9	1.73	3.68
10	18	-	481.5	250.2	115.6	1.92	4.17
11	20	-	557.8	268.5	121.2	2.08	4.60

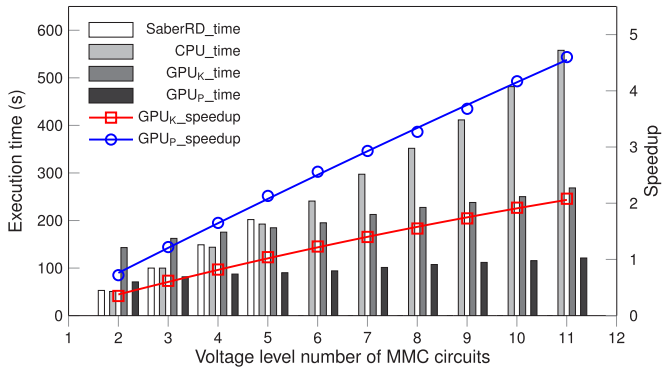


Fig. 20. One-phase nonlinear physics-based MMC circuit execution time and speedup comparison.

TABLE V
CPU AND GPU EXECUTION TIMES OF THREE-PHASE NONLINEAR PHYSICS-BASED MMC CIRCUIT

N_L	N_{SM}	Execution time (s)				Speedup		
		CPU	GPU$_K$	GPU$_P$	2GPU$_P$	GPU$_K$	GPU$_P$	2GPU$_P$
2	6	150.8	172.2	82.4	79.9	0.88	1.83	1.89
3	12	285.9	185.9	93.4	83.2	1.54	3.06	3.44
4	18	423.5	206.7	103.2	88.7	2.05	4.10	4.77
5	24	600.3	227.2	115.9	95.8	2.64	5.18	6.27
6	30	774.1	251.5	129.2	101.5	3.08	5.99	7.63
7	36	991.1	286.2	148.4	109.2	3.46	6.68	9.08
8	42	1249.2	302.3	164.2	117.8	4.13	7.61	10.60
9	48	1495.4	322.9	182.3	127.2	4.63	8.20	11.76
10	54	1782.4	357.2	202.5	134.4	4.99	8.80	13.26
11	60	2137.3	381.1	223.2	145.7	5.61	9.58	14.67

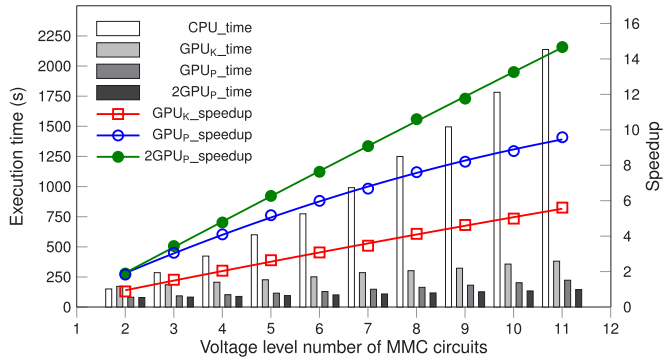


Fig. 21. Three-phase nonlinear physics-based MMC circuit execution times and speedup comparison.

since the run time of the CPU program was close to that of SaberRD according to existing numbers, the CPU results were used to calculate speedup replacing the incomplete results of SaberRD for circuits with more than eight SMs. As shown in Fig. 20, although the speedup rates of GPUs are less than 1 when the levels of MMC circuits are lower, up to 4 for GPU$_K$ and 2 for GPU$_P$, they increase steadily along with the circuit scale, which approach 2.08 for GPU$_K$ and 4.60 for GPU$_P$. Benefitting from the higher clock and memory data rate listed in Table III, GPU$_P$ doubles the speedup of GPU$_K$.

Although some speedups are shown in the simulations of single-phase MMC circuits, they are not representative of the full computational capability of GPUs for MMC simulation. In order to feed more data to the GPUs to realize their compute potential, 3-phase physics-based MMC circuits from 2-level to 11-level are simulated for 100 ms with the variable time-stepping method, whose execution time is listed in Table V. For three-phase systems, SaberRD is difficult to converge beyond two-level MMC circuit, and therefore, it is absent from the comparison. In addition, two Pascal GPU (GTX 1080) are involved in the competition since there is sufficient compute load to be processed. When the 3-phase MMC circuit reaches 11-level with 60 SMs, the acceleration of GPUs is obvious, which is 5.61 for GPU$_K$ and 9.58 for GPU$_P$. Moreover, the 2-GPU$_P$ platform obtains the speedup of up to 14.67, which almost triples the performance of Kepler GPU, as shown in Fig. 21. The specific structure of MMC circuit consisting of upper and lower arms simplifies the task management and the computational load balancing of 2-GPU$_P$ platform, and only the data of one common node need to be exchanged between the two MMC arms.

TABLE VI
CPU AND GPU EXECUTION TIMES OF THREE-PHASE BEHAVIOR-BASED MMC CIRCUIT

N_L	N_{SM}	N_{IGBT}	Execution time (s)				Speedup		
			CPU	GPU$_K$	GPU$_P$	2GPU$_P$	GPU$_K$	GPU$_P$	2GPU$_P$
11	60	120	37.2	35.5	14.1	14.3	1.05	2.64	2.60
21	120	240	67.4	35.9	14.4	14.5	1.88	4.68	4.65
41	240	480	129.2	36.2	14.5	14.6	3.57	8.91	8.85
51	300	600	156.0	36.6	14.6	14.7	4.26	10.68	10.61
81	480	960	245.8	38.2	14.9	14.9	6.43	16.50	16.50
101	600	1200	302.8	40.9	16.4	15.1	7.40	18.46	20.05
126	750	1500	376.2	41.9	16.8	15.4	8.98	22.39	24.43
151	900	1800	453.2	43.1	17.2	15.8	10.52	26.35	28.68
176	1050	2100	526.2	46.2	18.8	16.8	11.39	27.99	31.32
201	1200	2400	595.6	47.3	19.0	17.1	12.59	31.35	34.83
251	1500	3000	755.2	49.7	20.1	17.6	15.20	37.57	42.91
301	1800	3600	894.5	53.2	21.4	18.5	16.81	41.80	48.35
401	2400	4800	1188.2	56.1	23.2	19.9	21.18	51.22	59.71
501	3000	6000	1495.6	62.5	25.5	21.6	23.93	58.65	69.24

Since the Jacobian matrix is prone to singularity as the voltage levels increase, the nonlinear physics-based MMC circuits are hard to converge; therefore, the MMC circuits larger than 11-level are simulated with behavior-based model from 11-level to 501-level for 1 s with a 10 μs fixed time-step method, whose

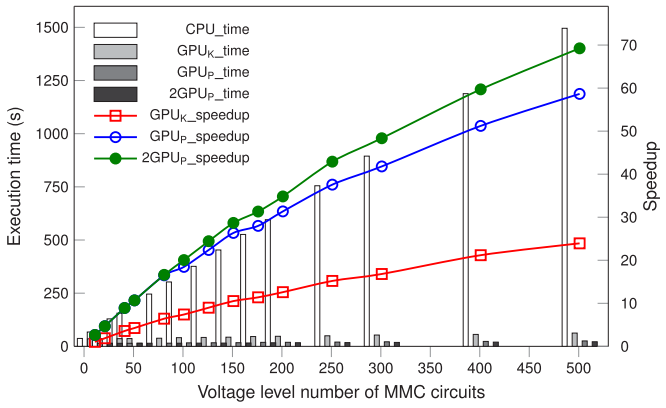


Fig. 22. Threee-phase behavior-based MMC circuit execution times and speedup comparison.

execution times are listed in Table VI and plotted in Fig. 22. When the circuit size expands to 501-level containing 3000 SMs, all simulations using GPUs obtain significant gain due to the simplicity of the behavior-based component models relative to the physics-based models. Since each IGBT of whole arm is treated as a Thévenin equivalent circuit in the behavior-based model, the computation of all SMs could be decoupled and processed independently without iteration after the arm current was calculated, which perfectly fit the massively parallel architecture of the GPUs. When circuit scale is small (less than 480 SMs), the 2-GPU$_P$ platform performs with slightly lower efficiency than 1-GPU$_P$ platform because one Pascal GPU can fulfil the computational requirement of the simulation for small systems, whereas data exchange between the two GPUs produce additional latency. At 501-level, the 1-GPU$_P$ and 2-GPU$_P$ compute platforms achieve speedups of 58 and 70, respectively.

V. Conclusion

System-level and device-level simulations provide different focus in power electronic circuit simulation. Adopting nonlinear physics-based device-level models to construct large-scale power electronic converter circuits gives an opportunity to study detailed device interactions while simultaneously addressing the challenge of large-scale system solution. The cost is to manage the high computational burden brought about by the complex system model. CPU-based sequential implementation results in higher execution times, with the growth of converter levels, which makes the simulation impractical. With the development of GPU-based massively parallel platforms, this computational challenge can be addressed; however, it requires a mapping of the nonlinear system model and its solution algorithm to conform to the massively parallel architecture, and efficient parallel programming. This paper presented a massively parallel implementation of device-level nonlinear physics-based IGBT and power diode models, and the corresponding numerical solvers on GPUs. Significant speed-ups were observed using both the physics-based as well as behavioral models of the devices. To handle the growing dimension of the Jacobian matrix with the increase in converter levels, the proposed block computation using partial LU decomposition increases the system level parallelism, which can help solve large power electronic circuits containing repetitive structures. The PCVTSM increases the simulation efficiency by arranging computational resources according to the required tasks, which can benefit a wide range of circuit simulation applications. The main limitation of using nonlinear device-level models for high-level MMC circuit simulation comes from the ill conditioning of the Jacobian matrix. One approach is to increase the precision of calculation with the improvement of hardware, and there are ways to improve the convergence characteristics of the Newton–Raphson method. Future research will focus on these directions along with an exploration of utilizing GPUs for the simulation of large-scale multiterminal dc grids.

Appendix

TABLE VII
Single-Phase Five-Level Physics-Based MMC Circuit Specifications

System parameters					
Arm inductance L_a	5 mH		Load inductance L_L		5 mH
Load Resistance R_L	4.6 Ω		SM capacitance C_m		4 mF
DC voltage V_{dc}	1000 V		Gate resistance R_g		100 Ω
System frequency f_s	60 Hz		Carrier signal frequency f_c		2500 Hz
Simulation time t_s	100 ms		Initial time-step size Δt_i		1 ns
Diode parameters					
I_S	10^{-14} A	τ	5 μs	T_M	5 μs
V_T	0.0259 V	m	0.5	C_{j0}	1 nF
V_J	0.7 V	I_{SE}	10^{-22}	R_c	10^{-3} Ω
IGBT parameters					
A $= 0.1$ cm^2, AGD $= 0.05$ cm^2, $I_{sne} = 6. \times 10^{14}$ A, FLTD $= 1 \times 10^{-3}$ V, $V_{crit} = 0.6$ V, $\tau_{HL} = 0.6$ s, $K_f = 1.0$, $K_p = 0.38$ A/V, $\theta = 0.02$ V^{-1}, $V_t = 4.7$ V, $W_B = 0.009$ cm, BVf $= 1$, BVn $= 4$, $C_{gs} = 6.2 \times 10^{-10}$ F, $C_{oxd} = 1.75 \times 10^{-9}$, NB $= 2.0 \times 10^{14}$ cm^{-3}.					

TABLE VIII
3-Phase 11-Level Physics-Based MMC Circuit Specifications

System parameters			
Arm inductance L_a	5 mH	Load inductance L_L	5 mH
Load Resistance R_L	20 Ω	SM capacitance C_m	4 mF
DC voltage V_{dc}	2000 V	Gate resistance R_g	100 Ω
AC voltage V_s	2000 V	Rated power P_{rated}	600 kW
System frequency f_s	60 Hz	Carrier signal frequency f_c	2500 Hz
Simulation time t_s	100 ms	Initial time-step size Δt_i	1 ns

TABLE IX
3-Phase 201-Level Behavior-Based MMC Circuit Specifications

System parameters			
Arm inductance L_a	150 mH	Load inductance L_L	3 mH
Load Resistance R_L	5 Ω	SM capacitance C_m	4 mF
DC voltage V_{dc}	38 kV	Gate resistance R_g	100 Ω
System frequency f_s	60 Hz	Carrier signal frequency f_c	2500 Hz
Simulation time t_s	0.5 s	Fixed time-step size Δt_f	10 μs

REFERENCES

[1] A. Ammous *et al.*, "Electrothermal modeling of IGBTs: Application to short-circuit conditions," *IEEE Trans. Power Electron.*, vol. 15, no. 4, pp. 778–790, Jul. 2000.

[2] W. Zhou, X. Zhong, and K. Sheng, "High temperature stability and the performance degradation of SiC MOSFETs," *IEEE Trans. Power Electron.*, vol. 29, no. 5, pp. 2329–2337, May 2014.

[3] A. Subbiah and O. Wasynczuk, "Computationally efficient simulation of high-frequency transients in power electronic circuits," *IEEE Trans. Power Electron.*, vol. 31, no. 9, pp. 6351–6361, Sep. 2016.

[4] "Saber industry standard for multi-domain and mixed-signal simulation," Synopsys Inc., Mountain View, CA, USA, 2007.

[5] L. Johnson, "Saber-MATLAB integrations: Enabling virtual HW/SW co-verification," Synopsys, Inc., Mountain View, CA, USA, Jun. 2004

[6] "PSPICE analog and mixed signal simulation," Cadence Design Systems, Inc., San Jose, CA, USA, 2000.

[7] O. A. Ahmed and J. A. M. Bleijs, "Pspice and simulink co-simulation for high efficiency DC-DC converter using SLPS interface software," in *Proc. 5th IET Int. Conf. Power Electron. Mach. Drives*, Brighton, U.K., 2010, pp. 1–6.

[8] "Saber accelerates robust design," Synopsys, Inc., Mountain View, CA, USA, vol. 5, Jun. 2007

[9] A. R. Hefner and D. M. Diebolt, "An experimentally verifed IGBT model implemented in the Saber circuit simulator," *IEEE Trans. Power Electron.*, vol. 9, no. 5, pp. 532–542, Sep. 1994.

[10] A. R. Hefner, "Modeling buffer layer IGBTs for circuit simulation," *IEEE Trans. Power Electron.*, vol. 10, no. 2, pp. 111–123, Mar. 1995.

[11] 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.

[12] *CUBLAS LIBRARY User Guide*, NVIDIA Corp., Sep. 2016.

[13] *CUFFT LIBRARY User's Guide*, NVIDIA Corp., Sep. 2016.

[14] P. Bailey, J. Myre, S. D. C. Walsh, D. J. Lilha, and M. O. Saar, "Accelerating lattice Boltzmann fluid flow simulations using graphics processors," in *Proc. 2009 Int. Conf. Parallel Process.*, Vienna, Austria, Sep. 2009, pp. 550–557.

[15] M. Smelyanskiy, D. Holmes, J. Chhugani, and A. Larson, "Mapping high-fidelity volume rendering for medical imaging to CPU, GPU and many-core architectures," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1563–1570, Aug. 2010.

[16] J. D. Owens, M. Houston, D. Luebke, and S. Green, "GPU Computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[17] M. Rietmann, P. Messmer, T. Nessen-Meyer, and D. Peter, "Forward and adjoint simulations of seismic wave propagation on emerging large-scale GPU architectures," in *Proc. 2012 Int. Conf. High Perform. Comput., Storage Anal.*, Salt Lake City, UT, USA, Nov. 2012, pp. 1–11.

[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] Z. Zhou and V. Dinavahi, "Parallel massive-thread electromagnetic transient simulation on GPU," *IEEE Trans. Power Del.*, vol. 29, no. 3, pp. 1045–1053, Jun. 2014.

[20] H. Karimipour and V. Dinavahi, "Extended Kalman filter-based parallel dynamic state estimation," *IEEE Trans. Smart Grid*, vol. 6, no. 3, pp. 1539–1549, May 2015.

[21] H. Karimipour and V. Dinavahi, "Parallel relaxation-based joint dynamic state estimation of large-scale power systems," *IET Gener., Transmiss., Distrib.*, vol. 10, no. 2, pp. 452–459, Feb. 2016.

[22] P. Liu and V. Dinavahi, "Finite-difference relaxation for parallel computation of ionized field of HVDC lines," *IEEE Trans. Power Deliv.*, pp. 1–10, 2017.

[23] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units," *IEEE Power Tech*, Lausanne, Switzerland, 2007, pp. 731–736.

[24] P. O. Lauritzen and C. L. Ma, "A simple diode model with reverse recovery," *IEEE Trans. Power Electron.*, vol. 6, no. 2, pp. 188–191, Apr. 1991

[25] *NVIDIA's Next Generation CUDATM Compute Architecture: KeplerTM GK110* NVIDIA Corp., Santa Clara, CA, USA, 2012.

[26] *Whitepaper NVIDIA GeForce GTX 1080*, NVIDIA Corp., Santa Clara, CA, USA, 2016.

[27] *NVIDIA CUDA C Programming Guide Version 7.0*, NVIDIA Corp., Santa Clara, CA, USA, Mar. 2015.

[28] C. L. Ma and P. O. Lauritzen, "A simple power diode model with forward and reverse recovery," *IEEE Trans. Power Electron.*, vol. 8, no. 4, pp. 342–346, Oct. 1993.

[29] G. T. Oziemkiewicz, "Implementation and development of the NIST IGBT model in a SPICE-based commercial circuit simulator," M.Sc. thesis, Dept. Elect. Comput. Eng., Univ. Florida, Gainesville, FL, USA, 1995.

[30] S. Debnath, J. Qin, B. Bahrani, M. Saeedifard, and P. Barbosa, "Operation, control, and applications of the modular multilevel converter: A review, " *IEEE Trans. Power Electron.*, vol. 30, no. 1, pp. 37–53, Jan. 2015.

[31] D. C. Ludois and G. Venkataramanan, "Simplified terminal behavioral model for a modular multilevel converter," *IEEE Trans. Power Electron.*, vol. 29, no. 4, pp. 1622–1631, Apr. 2014.

[32] H. Peng, M. Hagiwara, and H. Akagi, "Modeling and analysis of switching-ripple voltage on the DC link between a diode rectifier and a modular multilevel cascade inverter (MMCI)," *IEEE Trans. Power Electron.*, vol. 28, no. 1, pp. 75–84, Jan. 2013.

[33] Z. Shen and V. Dinavahi "Real-time device-level transient electrothermal model for modular multilevel converter on FPGA," *IEEE Trans. Power Electron.*, vol. 31, no. 9, pp. 6155–6168, Sep. 2016.

[34] H. Saad *et al.*, "Dynamic averaged and simplified models for MMC-Based HVDC transmission systems," *IEEE Trans. Power Del.*, vol. 28, no. 3, pp. 1723–1730, Apr. 2013.

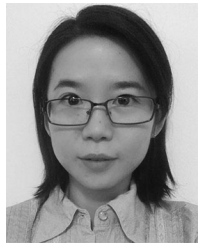
[35] M. Hagiwara and H. Akagi, "Control and experiment of pulse width-modulated modular multilevel converters," *IEEE Trans. Power Electron.*, vol. 24, no. 7, pp. 1737–1746, Jul. 2009.

[36] H. Akagi, S. Inoue, and T. Yoshii, "Control and Performance of a transformerless cascade PWM STATCOM with star configuration," *IEEE Trans. Ind. Appl.*, vol. 43, no. 4, pp. 1041–1049, Jul./Aug. 2007.

[37] G. P. Adam, O. Anaya-Lara, G. M. Burt, D. Telford, B. W. Williams, and J. R. Mcdonald, "Modular multilevel inverter: Pulse width modulation and capacitor balancing technique," *IET Power Electron.*, vol. 3, no. 5, pp. 702–715, Sep. 2010.

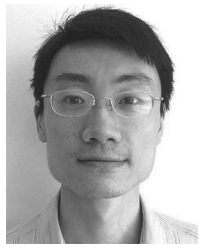
[38] C. Gear, "Simultaneous numerical solution of differential-algebraic equations," *IEEE Trans. Circuit Theory*, vol. 18, no. 1, pp. 89–95, Jan. 1971.

[39] Ó. Jiménez, Ó. Luca, I. Urriza, L. A. Barragan, D. Navarro, and V. Dinavahi, "Implementation of an FPGA-Based online hardware-in-the-loop emulator using high-level synthesis tools for resonant power converters applied to induction heating appliances," *IEEE Trans. Ind. Electron.*, vol. 62, no. 4, pp. 2206–2214, Apr. 2015.



Shenhao Yan (M'17) received the B.Eng. degree in electrical engineering and automation from Zhejiang University, Hangzhou, China, in 2012, and the M.Sc. degree in energy systems from the Department of Electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2016.

She is currently a Professional Electrical Engineer focusing on powertrain at Tesla, Inc., Shanghai, China. Her research interests include power electronics device modeling, device-level simulation, and massively parallel programming.



Zhiyin Zhou (S'12) received the B.Sc. degree in electronic science and engineering in 2000 from Nanjing University, Nanjing, China, and the M.Sc. degree in electrical and computer engineering in 2012 from the University of Alberta, Edmonton, AB, Canada, where he is currently working toward the Ph.D. degree in electrical and computer engineering.

His research interests include large-scale parallel and distributed computing, massively parallel programming, device-level transient analysis, HVdc power system simulation, and electromagnetic transient studies.

Venkata Dinavahi (SM'08) received the Ph.D. degree in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 2000.

He is currently a Professor in the Department of Electrical and Computer Engineering, 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.