This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

Hybrid Parallel-in-Time-and-Space Transient Stability Simulation of Large-Scale AC/DC Grids

Tianshi Cheng, *Graduate Student Member, IEEE,* Ning Lin, *Member, IEEE,* and Venkata Dinavahi, *Fellow, IEEE*

***Abstract*—The increasing complexity of modern AC/DC power systems poses a significant challenge to a fast solution of large-scale transient stability simulation problems. This paper proposes the hybrid parallel-in-time-and-space (PiT+PiS) transient simulation on the CPU-GPU platform to thoroughly exploit the parallelism from time and spatial perspectives, thereby fully utilizing parallel processing hardware. The respective electromechanical and electromagnetic aspects of the AC and DC grids demand a combination of transient stability (TS) simulation and electromagnetic transient (EMT) simulation to reflect both system-level and equipment-level transients. The TS simulation is performed on GPUs in the co-simulation, while the Parareal parallel-in-time (PiT) scheduling and EMT simulation are conducted on CPUs. Therefore, the heterogeneous CPU-GPU scheme can utilize asynchronous computing features to offset the data transfer latency between different processors. Higher scalability and extensibility than GPU-only dynamic parallelism design is achieved by utilizing concurrent GPU streams for coarse-grid and fine-grid computation. A synthetic AC/DC grid based on IEEE-118 Bus and CIGRÉ DCS2 systems showed a good accuracy compared to commercial TSAT software, and a speedup of 165 is achieved with 48 IEEE-118 Bus systems and 192 201-Level detail-modeled MMCs. Furthermore, the proposed method is also applicable to multi-GPU implementation where it demonstrates decent efficacy.**

***Index Terms*—Electromagnetic transients, graphical processors, high-voltage direct current, multi-core processors, multi-terminal DC grid, parallel-in-space, parallel-in-time, parallel processing, synchronous generator, transient stability.**

NOMENCLATURE

CPU	Central Processing Unit
GPU	Graphical Processing Unit
DAE	Differential Algebraic Equation
EMT	Electromagnetic Transient
HBSM	Half Bridge Sub-Module
HVDC	High Voltage Direct Current
IGBT	Insulated Gate Bipolar Transistor
IVP	Initial Value Problem
ODE	Ordinary Differential Equation
PiT	Parallel-in-Time
PiS	Parallel-in-Space
TLM	transmission line models
TS	Transient Stability
MTDC	Multi-Terminal Direct Current
MMC	Modular Multilevel Converter
RMS	Root Mean Square
SIMT	Single Instruction Multiple Threads
SM	Streaming Multiprocessor

This work is supported by the Natural Science and Engineering Research Council of Canada (NSERC).

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta T6G 2V4, Canada. Email: tcheng1@ualberta.ca, ning3@ualberta.ca, dinavahi@ualberta.ca.

I. INTRODUCTION

Modern power systems are increasingly complex due to the continuous integration of power electronic facilities such as the high voltage direct current (HVDC) links into transmission and distribution networks. Many HVDC projects are constructed or planned worldwide to integrate more clean energy from wind farms and PV stations, such as Changji-GuQuan UHVDC project in China [1], Dolwin 5 project in Europe [2], and TransWest project in the USA [3]. Using a typical step size of a few milliseconds, the transient stability (TS) analysis plays an important role in the planning, design, and operation of a modern power grid from a system point of view. It, however, is a positive-sequence-based analytical method that naturally falls short of complicated electromagnetic transient (EMT) details of power electronic devices in the microsecond-level or below. The TS-EMT co-simulation methodology which properly features both system-level and equipment-level power system phenomena is favored in hybrid AC/DC grid study. A consequently incurred rise of computational burden may extend the simulation duration, especially considering that a dramatic expansion of a future AC/DC power system as a result of incorporating more components is expected.

To handle the increasing scale and complexities, new acceleration techniques for TS and EMT simulation programs are desired. TS acceleration methods based on parallel processing algorithms for multi-core CPUs and many-core GPUs have shown a decent efficiency and have been well investigated in AC power grid studies [4]–[7] , and heterogeneous CPU-GPU computing architecture for AC-DC grid TS-EMT co-simulation has recently been proposed [8]–[10], while the threads concurrency of these methods is dominantly contributed by parallel-in-space (PiS) strategies. The parallel-in-time (PiT) solutions were also investigated from a different perspective of parallel processing [11]. The very early version of PiT for solving TS problems was mainly based on Jacobi-decomposition [12], [13], which aimed to solve multiple continuous steps iteratively so that the parallelism was achieved by assigning a single step to parallel threads. In the 1990s, most results were obtained from the virtual parallel machine because of a scarcity of multi-core CPUs, which limited further explorations in this area. The Parareal algorithm has emerged in many research areas [14], [15] where it exhibited efficacy [16]. It was introduced to solve TS simulation problems by decomposing the initial value problem into many sub-intervals [17], and has a better efficiency compared to its predecessors. These works mainly focused on PiT algorithms and potential comprehensive parallelism by considering PiS

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

methods simultaneously is yet to be carried out. For example, a four times speedup was obtained in computing the IEEE 39-bus system [17] with 470 cores, and even a huge number of cores were used to solve a large-scale power system, the parallel efficiency was below 20% [18], which is still not as satisfactory as PiS methods.

This paper proposes a hybrid parallel-in-time-and-space (PiT+PiS) AC/DC TS-EMT co-simulation method which thoroughly exploits parallelism to expedite the simulation of modern power systems on many-core GPUs. It demonstrates: (1) High level of parallelism: PiT+PiS can achieve a higher speedup and efficiency than PiS or PiT-only methods; (2) Efficient CPU-GPU communication implementation: utilizing asynchronous unified CUDA® memory to offset CPU-GPU communication overheads; (3) High scalability: using asynchronous multi-stream design to handle subsystems and schedule the workload on multiple GPUs/CPUs; (4) Multi-fold hybrid co-simulation: using applying the PiT+PiS method to TS-EMT co-simulation on CPU and GPU, the impacts of AC/DC system can be analyzed in a fast and accurate way.

This paper is organized as follows: Section II introduces the multi-mass synchronous machine model, MMC model, and theoretical speedup analysis of PiT and PiT+PiS methods; Section III introduces the implementations of hybrid CPU-GPU PiT+PiS algorithm for AC/DC co-simulation; Section IV presents the case studies and performance comparison; Section V is the Conclusion.

II. System Modeling and Parallel-in-Time Simulation

A. Multi-Mass Torsional Shaft Generator Model

The generator equations comprise of three components, i.e., the synchronous machine, the mechanical multi-mass torsional shaft, and the control system, which together form a 17th-order model.

As shown in Fig. 1, the synchronous machine includes two windings on d-axis and two damping windings on q-axis, which is classified as Model 2.2 in IEEE Std 1110-2019 [19]. The machine model can be expressed by differential equations as [20]:

$$\begin{aligned}\dot{\Psi}_{fd}(t) &= \omega_R \cdot [e_{fd}(t) - R_{fd} i_{fd}(t)] \\ \dot{\Psi}_{1d}(t) &= -\omega_R \cdot R_{1d} i_{1d}(t) \\ \dot{\Psi}_{1q}(t) &= -\omega_R \cdot R_{1q} i_{1q}(t) \\ \dot{\Psi}_{2q}(t) &= -\omega_R \cdot R_{2q} i_{2q}(t),\end{aligned} \tag{1}$$

where Ψ_{fd} and e_{fd} are flux linkage and field voltage of the field winding, Ψ_{1d}, Ψ_{1q}, Ψ_{2q} are the flux linkages of direct and quadrature axis windings; R_{1d}, R_{1q} and R_{2q} are the resistance of direct and quadrature axis windings; ω_R is the rated rotating speed. All the quantities are using the per unit system defined in [20] except the time t since the time domain simulation use seconds for the time unit. The currents in the dq frame are coupled with external power system equations as the stator currents must be obtained. To simplify the computation, an iterative method is used to handle the coupling [6].

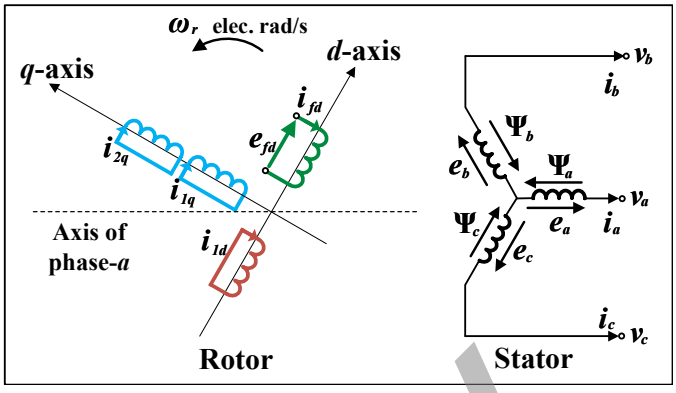

Fig. 1: Rotor and stator circuits of a Model 2.2 synchronous machine.

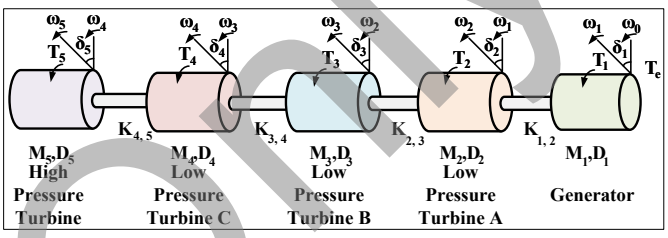

Fig. 2: Multi-mass model for mechanical side of the generator.

The mechanical shaft, on the other hand, provides basic rotor equations for the machine:

$$\begin{aligned}\dot{\delta}(t) &= \omega_R \cdot \Delta\omega(t), \\ \Delta\dot{\omega}(t) &= \frac{1}{2H}\left[T_m(t) - T_e(t) - D \cdot \Delta\omega(t)\right].\end{aligned} \tag{2}$$

Since the synchronous generator is connected to the turbine, the four-mass-turbine shaft model is used, as shown in Fig. 2, where δ_n means the relative rotor angle of each turbine, and specifically, δ_1 is the generator rotor angle. The stiffness coefficient between two neighboring masses is represented by parameter K, e.g., $K_{4,5}$ represents the coefficient between Mass5 and Mass4. T_n, D_n and H_n refer to the torque, damping factor and inertia constant of each torsional shaft, respectively, as given by:

$$\begin{aligned}&\Delta\dot{\omega}_n(t) = \\ &\frac{\left[T_n + K_n^{n+1}(\delta_{n+1}(t) - \delta_n(t)) - K_{n-1}^{n}(\delta_n(t) - \delta_{n-1}(t)) - D_n \Delta\omega_n(t)\right]}{2H_n}, \\ &\dot{\delta_n}(t) = \omega_R \cdot \Delta\omega_{n(t)}, n = 2, 3, 4, 5.\end{aligned} \tag{3}$$

The control system includes PSS, excitation, and AVR systems, which is classified as ST1A model in [21]:

$$\begin{aligned}\dot{v}_1(t) &= \frac{1}{\tau_R} \cdot \left[|\boldsymbol{v}_{dq}| - v_1(t)\right] \\ \dot{v}_2(t) &= K_{stab} \cdot \Delta\dot{\omega}(t) - \frac{1}{\tau_\omega} v_2(t) \\ \dot{v}_3(t) &= \frac{1}{\tau_2} \cdot \left[\tau_1 \dot{v}_2(t) + v_2(t) - v_3(t)\right],\end{aligned} \tag{4}$$

where τ_R, τ_ω, τ_1, τ_2, and K_{stab} are constants, v_1, v_2, v_3 are state variables. The generator equations and power system

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

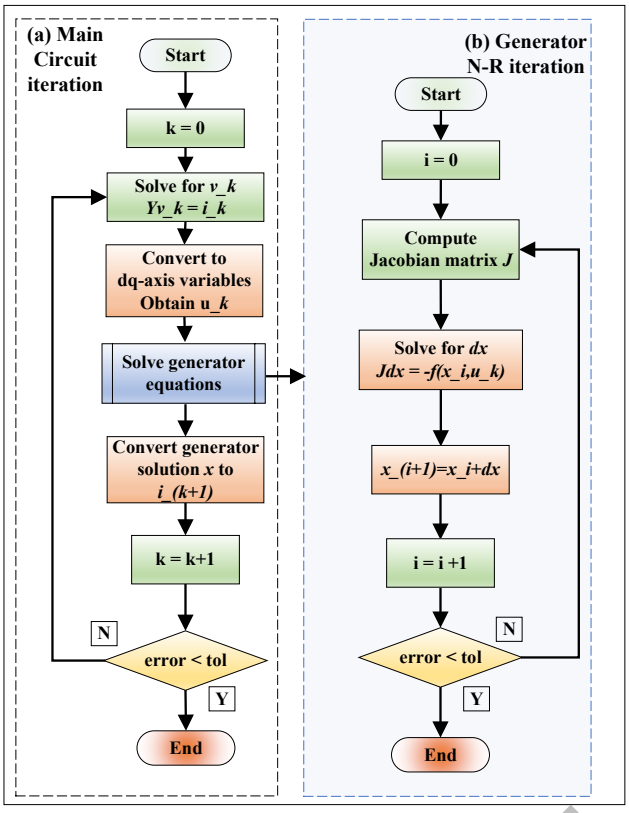

Fig. 3: Transient stability simulation process: (a) loop for solving voltages in main circuit; (b) loop for solving generater state variables.

network equations constitute the differential-algebraic equations (DAEs) of transient stability simulation, which can be expressed as

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}, \boldsymbol{u}), \quad g(\boldsymbol{x}, \boldsymbol{u}) = 0, \\ \boldsymbol{x}_0 &= \boldsymbol{x}\left(t_0\right), \quad \boldsymbol{i}_0 = \boldsymbol{i}\left(t_0\right), \end{aligned} \tag{5}$$

where $\boldsymbol{x} = \{\Psi_{fd}, \Psi_{1d}, \Psi_{1q}, \Psi_{2q}, \delta_1, \Delta\omega_1, \delta_2, \Delta\omega_2, \delta_3, \Delta\omega_3, \delta_4, \Delta\omega_4, \delta_5, \Delta\omega_5, v_1, v_2, v_3\}$ is the generator state vector , $\boldsymbol{u} = \{i_{1d}, i_{1q}, i_{2q}, e_{fd}, i_{fd}, \boldsymbol{v}_{dq}\}$ is the vector of system inputs; f is the vector function of equations (1)-(4); g is the algebraic equation to solve the input vector $\boldsymbol{u}$.

In addition, the stator equations [20], [22] are necessary to solve the interface voltages and currents along with the external grid, making the simulation a DAE problem. The generator's variables are in d- and q-axis so Park transformation is involved in solving the DAE [23], which makes it a nonlinear problem. As shown in Fig. 2, a partitioned iterative method [6] is used to solve the DAE, which decomposes generators from the main circuits.

With implicit Trapezoidal rule, an individual generator has the following discretized equation

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{1}{2}h(\boldsymbol{f}(\boldsymbol{x}_{n+1}, \boldsymbol{u}_{n+1}) + \boldsymbol{f}(\boldsymbol{x}_n, \boldsymbol{u}_n)), \tag{6}$$

where n indicates the number of discrete steps, h indicates the time-step. Since solving such an implicit equation requires f_{n+1} at x_{n+1} and u_{n+1}, implicit methods requires the Jacobian matrix $\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$. The generator equation can be expressed by a state-space form:

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = A\boldsymbol{x} + B\boldsymbol{u}, \tag{7}$$

where A and B are coefficient matrices of Equation (1-4). Then The equation to solve x_{n+1} can be expressed by

$$(E - \frac{h}{2}A)\boldsymbol{x}_{n+1} - (E + \frac{h}{2}A)x_n + \frac{h}{2}hB\boldsymbol{u}_{n+1} = 0 \tag{8}$$

where $(E - \frac{h}{2}A)$ is the Jacobian matrix. If f_{n+1} is nonlinear, A is no longer constant so that the Newton-Raphson iteration is required. Otherwise, it can be solved without local iteration [24].

B. MMC Model

To model the DC power systems, an equivalent-circuit-based MMC model for EMT simulation is adopted in this work. For the concurrent implementation of the half-bridge submodules (HBSMs), an artificial delay is assumed between the submodules and the main circuit to decouple each submodule and significantly reduce the number of circuit nodes while details are still preserved, as shown in Fig. 4 (a). The ON and OFF states of IGBTs and diodes are represented by low- and high-impedance.

Each submodule is solved independently, whichindicates that the MMC computation is transformed into solving a number of 2x2 matrix equations for v_{sm} and v_c of each SM. At an arbitrary time instant n, represented by the capacitor voltage v_c and submodule port voltage v_{sm}, the 2 unknown nodes can be solved by the following discretized equations:

$$\begin{aligned} &\begin{bmatrix} g_{sw1} + g_{sw2} & -g_{sw1} \\ -g_{sw1} & g_{sw2} \end{bmatrix} \begin{bmatrix} v_c(n) \\ v_{sm}(n) \end{bmatrix} = \begin{bmatrix} i_{ceq}(n) \\ i_{arm}(n-1) \end{bmatrix}, \\ &i_{ceq}(n) = v_c(n-1)G_{ceq} + i_c(n-1), \end{aligned} \tag{9}$$

where G_{ceq}, i_{ceq} denote the capacitor equivalent admittance and the companion current, $g_{sw1,2}$ denote the equivalent admittance of the IGBT/diode pair, and i_c is the capacitor current. The g_{sw1} and g_{sw2} are determined by

$$g_{sw} = v_g G_{on} + (1 - v_g)G_{off}, v_g \in \{0, 1\} \tag{10}$$

where the gate signal v_g has been converted into a binary value in the simulation, G_{on} and G_{off} are the turn-on conductance and turn-off conductance. The SM circuit can also use more detailed device-level IGBT models instead of ideal two-state switches. Since the equivalent circuit MMC model preserves individual submodule, equipment-level transients such as switching harmonics and capacitor energy balancing can be studied in EMT simulation [25].

Since the HBSM topologies under different switching states are known according to the submodule port current and control signals, the inverted SM admittance matrices are cached to speed up the computation. After solving v_{sm} of each SM, all SMs can be lumped into a voltage source and it can be further simplified to a reduced Norton equivalent circuit by merging the arm-choke inductor as shown in Fig. 4 (b).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

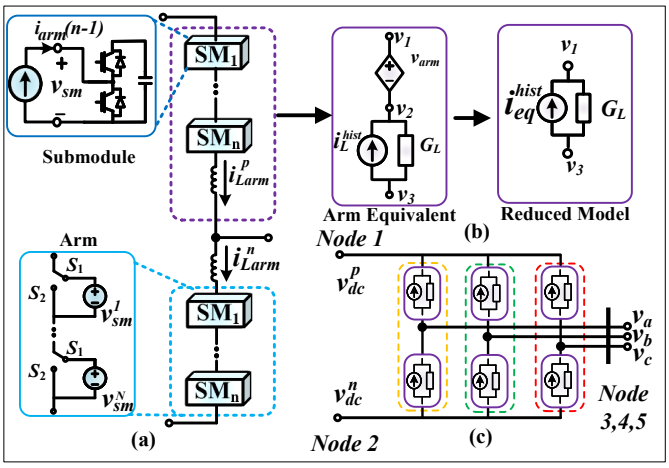

Fig. 4: Three-phase MMC model: (a) the MMC arm submodule branches are equivalent to a series of voltage sources; (b) the choke inductor can be merged to omit one internal node; (c) the final MMC equivalent circuit in the main circuit.

The nearest level modulation (NLM) is used as the MMC internal controller to maintain stable submodule capacitor voltages. The number of inserted submodules n_{ins} is determined by:

$$n_{ins} = round(\frac{n_{sm} v_{ref} + n_{sm}}{2}) \tag{11}$$

where n_{sm} is the SM number in one arm, v_{ref} is the reference voltage in per unit. After the n_{ins} is obtained, the lower-level voltage balancing controller will use it to generate gate signals for each SM and the voltage balancing is based on a widely used sorting strategy [26].

The MMC main circuit under nodal analysis is shown in Fig. 4(c). Compared with the submodules, it has a minimum of 5 nodes even after converting each arm to its Norton equivalent circuit form. Noticing that instant solution as it is with the submodules is not readily available, which causes an artificial delay between main circuit and submodules. The KLU method [27] is adopted to solve the main circuit matrix equation.

C. Parareal Algorithm

The Parareal algorithm decomposes a large time interval into many small sub-intervals and solves the corresponding differential equations in parallel, and therefore, it is considered as an iterative multi-shooting approach [28]. As the differential equations present an initial value problem (IVP), initialization is mandatory for the solution process, and a fast serial predictor is used to provide the initial conditions, which divides the problem into a serial coarse-grid and a parallel fine-grid.

The Parareal algorithm for an overall N intervals can be the expressed by following nonlinear system of equation:

$$\boldsymbol{E}(\boldsymbol{U}) := \begin{cases} \boldsymbol{U}_1 - \boldsymbol{F}(T_1, T_0, \boldsymbol{U}_0) = 0, \\ \boldsymbol{U}_2 - \boldsymbol{F}(T_2, T_1, \boldsymbol{U}_1) = 0, \\ \quad \vdots \\ \boldsymbol{U}_N - \boldsymbol{F}(T_N, T_{N-1}, \boldsymbol{U}_{N-1}) = 0, \end{cases} \tag{12}$$

where $\boldsymbol{U}_0$ is a vector containing the known initial values, $\boldsymbol{U}_j (j = \{1, 2..., N\})$ are the final solution produced by a fine solution operator $\boldsymbol{F}(T_j, T_{j-1}, \boldsymbol{U}_{j-1})$; T_j is the time instant of the N sub-intervals. By applying Newton's method,

$$(\boldsymbol{U}^{(k)} - \boldsymbol{U}^{(k-1)}) \frac{d}{d\boldsymbol{U}} \boldsymbol{E}(\boldsymbol{U}^{(k-1)}) = \boldsymbol{E}(\boldsymbol{U}^{(k-1)}), \tag{13}$$

the following iterative equation for each $\boldsymbol{U}_j$ is obtained:

$$\begin{aligned} \boldsymbol{U}_j^{(k)} &= \boldsymbol{F}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k-1)}\right) \\ &+ \frac{\partial \boldsymbol{F}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k-1)}\right)}{\partial \boldsymbol{U}_{j-1}^{(k-1)}} \left(\boldsymbol{U}_{j-1}^{(k)} - \boldsymbol{U}_{j-1}^{(k-1)}\right), \end{aligned} \tag{14}$$

where $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{U}}$ is the derivative of operator function $\boldsymbol{F}$, written in a discrete form as:

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{U}} = \frac{\boldsymbol{F}^{(k)} - \boldsymbol{F}^{(k-1)}}{\boldsymbol{U}^{(k)} - \boldsymbol{U}^{(k-1)}}. \tag{15}$$

If $\boldsymbol{F}$ is used to obtain $\boldsymbol{F}^{(k)}$, it is identical to the normal sequential solution with $\boldsymbol{F}$. However, the Quasi-Newton method can be used to approximate the Jacobian, which can be naturally done by an operator $\boldsymbol{G}$ with a larger time-step, so that the derivative can be generated to make the PiT computing feasible:

$$\begin{aligned} \boldsymbol{F}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k)}\right) &\approx \boldsymbol{G}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k)}\right) \\ \boldsymbol{F}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k-1)}\right) &\approx \boldsymbol{G}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k-1)}\right). \end{aligned} \tag{16}$$

The discrete sparse time points processed by $\boldsymbol{G}$ at $\{T_1, T_2, ..., T_N\}$ constitute the coarse-grid; The discrete time points processed by $\boldsymbol{F}$ in $[T_1, T_N]$ constitute the fine-grid. Since the serial $\boldsymbol{G}$ produced the approximations for each $[T_{j-1}, T_j]$ sub-interval, $\boldsymbol{F}$ can execute them in parallel.

$\boldsymbol{G}$ can also be a faster integration method like Euler and backward Euler method while $\boldsymbol{F}$ is a more time-consuming method such as Trapezoidal and Runge-Kutta method. In this work, the Trapezoidal integration method is used, which gives

$$\begin{aligned} \boldsymbol{F}^k &= \boldsymbol{x}_n^k + \frac{1}{2} h_F (\boldsymbol{f}(\boldsymbol{x}_{n+1}^k, \boldsymbol{u}_{n+1}^k) + \boldsymbol{f}(\boldsymbol{x}_n^k, \boldsymbol{u}_n^k)), \\ \boldsymbol{G}^k &= \boldsymbol{x}_n^k + \frac{1}{2} h_G (\boldsymbol{f}(\boldsymbol{x}_{n+1}^k, \boldsymbol{u}_{n+1}^k) + \boldsymbol{f}(\boldsymbol{x}_n^k, \boldsymbol{u}_n^k)), \quad h_F < h_G, \end{aligned} \tag{17}$$

where $\boldsymbol{f}$ refers to aforementioned equations (1)-(4), h_F and h_G are the fine-grid time-step and coarse-grid time-step respectively. The only difference between $\boldsymbol{F}$ and $\boldsymbol{G}$ is the size of time-steps and they are the same Trapezoidal method. Substituting $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{U}}$ with the approximation, the following equation is obtained to conduct the predict-correct iteration between coarse and fine grids:

$$\begin{aligned} \boldsymbol{U}_j^{(k)} &= \boldsymbol{F}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k-1)}\right) \\ &+ \boldsymbol{G}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k)}\right) - \boldsymbol{G}\left(T_j, T_{j-1}, \boldsymbol{U}_{j-1}^{(k-1)}\right), \end{aligned} \tag{18}$$

which was proven to be a Quasi-Newton method in [29].

As shown in Fig. 5, the Parareal algorithm has four major progressions: (a) *Initialization*: the initial guess is generated

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

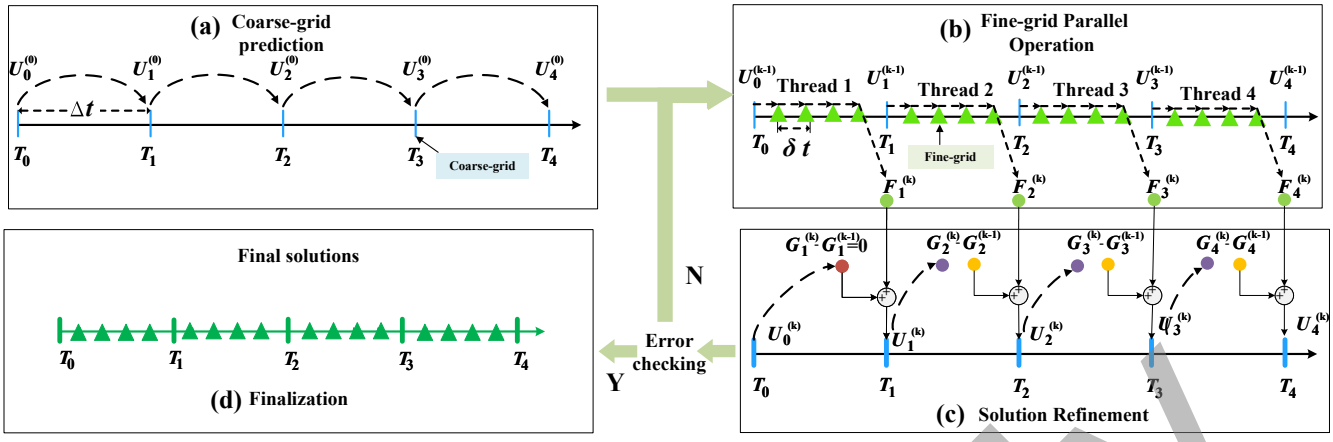

Fig. 5: Sequence of operations in the Parareal algorithm: (a) Initialize $\boldsymbol{U}_j^{(0)}$ which equals to $\boldsymbol{G}_j^{(0)}$; (b) Produce fine-grid solution $\boldsymbol{F}_j^k$; (c) Refine $\boldsymbol{U}_j^{(k)}$ with $\boldsymbol{U}_j^{(k)} = \boldsymbol{G}_j^{(k)} + \boldsymbol{F}_j^k - \boldsymbol{G}_j^{(k-1)}$, then continue (b) to start a new iteration.

from the coarse operator; (b) *Fine-grid Parallel Operation*: the fine-grid workers produce the fine-grid solutions concurrently from the initial values of coarse-grid; (c) *Solution Refinement*: the coarse-grid operator produces new predictions and correct the solutions at T_j using (18); (d) *Finalization*: If the error is smaller than tolerance, the fine-grid workers generate the final converged solutions. Otherwise, it continues to step (b) and continue the Parareal iterations.

D. Theoretical Speedup Analysis

Assuming a system with a fixed workload of $n \cdot w$, where n indicates the total fine-grid time-steps of the simulation and w is the single-step execution time of the DAE solution, and the same integration method for the fine-grid and coarse-grid, according to Amdahl's law [30], the speedup of the PiT algorithm can be expressed by

$$S_{pit}(p) = \frac{n \cdot w}{((I+1) \cdot p \cdot w + I \cdot m \cdot w)} = \frac{1}{((I+1)/m + I/p)}, \tag{19}$$

where S_{pit} is the speedup of Parareal, m is the steps of fine-grid sub-intervals, I is the iteration number and p is the number of PiT processors, $n = mp$. The number of parallel processors is related to the sizes of time-steps and time-windows, which means more processors will add difficulties to convergence, resulting in degraded speedup, the theoretical speedup limit is bounded to $min\{\frac{m}{I+1}, \frac{p}{I}\}$, which creates a tradeoff between convergence and parallelism [31].

The parallel efficiency E_{pit} of Parareal algorithm is

$$E_{pit}(p) = \frac{S_{pit}}{p} = \frac{1}{\frac{(I+1)p}{m} + I} < \frac{1}{I}, \tag{20}$$

which indicates that the maximum parallel efficiency is smaller than 50% due to the fact that $I \geq 2$.

In practice, m must be a large number to compensate overheads caused by coarse-grid and synchronizations. When $(I+1)/ >> I/p$ the theoretical speedup upper limit can be seen as $\frac{p}{I}$, while this limit is still significantly constrained by convergence. An alternative is to use a limited p (10 in this work) and a small time-window ($10ms \times 10 = 0.1s$ in this work).

The PiT+PiS method can retain spatial parallelism in each solution step of the PiT algorithm to improve the maximum parallel efficiency achieved by either method. The speedup of PiT+PiS is given by

$$\begin{aligned} S_{pit+pis}(p_1, p_2) &= \frac{S_{pis}(p_1) \cdot p_2 \cdot m \cdot w_{pis}}{(I+1) \cdot p_2 \cdot w_{pis} + I \cdot m \cdot w_{pis}} \\ &= S_{pis}(p_1) S_{pit}(p_2), \end{aligned} \tag{21}$$

where w_{pis} is the execution time with spatial parallel methods, p_1 is number of parallel processors for PiS, and p_2 is the number of parallel processors for PiT. (21) indicates that compared to PiT-only or PiS-only method, the PiT+PiS method can utilize more parallel processors to solve a problem with a fixed size. The parallel efficiency for PiT+PiS is expressed by

$$E_{pit+pis}(p_1, p_2) = E_{pis}(p_1) E_{pit}(p_2) < \frac{E_{pis}(p_1)}{I_{pit}(p_2)}. \tag{22}$$

where E is the parallel efficiency for each parallel method. In practice, $I(p_2) < I(p_1 p_2)$ is very common, and therefore, $E_{pit+pis}(p_1, p_2) > E_{pit}(p_1 p_2)$. Also, it is possible to achieve $E_{pit+pis}(p_1, p_2) > E_{pis}(p_1 p_2)$ when the PiS thread is saturated at p_1.

Assuming a system with 8 partitions. The parallel efficiency of a PiS method for this system workload is determined by

$$E_{pis}(p) = \begin{cases} \frac{-2}{35}p + \frac{-37}{35} & p <= 8 \\ 0.6 \times 8/p & p > 8 \end{cases} \tag{23}$$

which indicates that the system can utilize at most 8 threads for parallel computing.

The PiT method is assumed to have $m = 100, p = 10$, and $I(p) = 2 + int(p/10)$ which means that the iteration number increases by 1 when adding every 10 threads. Substituting these parameters with $p_1 = \{1, 2, ..., 200\}$ into (19-22) will gives the results shown in Fig. 6. Clearly, the PiT+PiS can achieve better performance compared to PiT-only and PiS-only methods. Also, the PiT+PiS method requires more threads so it may be suitable for GPUs with thousands of CUDA® cores.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

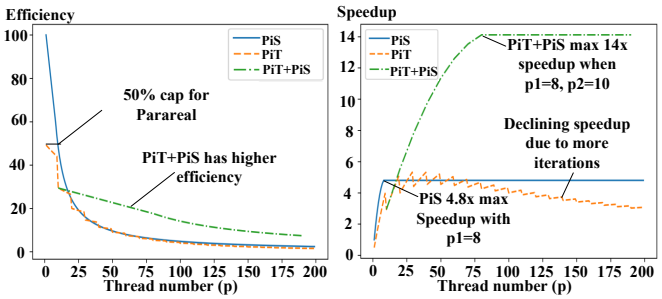

Fig. 6: Theoretical performance comparisons of PiS, PiT and PiT+PiS methods: (a) theoretical parallel efficiency; (b) theoretical speedup.

III. AC/DC Grid Parallel-in-Time-and-Space Co-Simulation

To establish an AC/DC PiT+PiS co-simulation program, a software hierarchy shown in Fig. 7 is proposed. Generally, the AC TS system solver and HVDC EMT system solver are developed independently and connected together via an interface.

A. GPU PiT+PiS Programming

The thousands of CUDA® cores of an NVIDIA® GPU are affiliated to dozens of streaming multiprocessors (SMs), which are responsible for scheduling instructions, as shown in Fig. 8. The frequency of GPU cores is much lower than many prevalent CPUs and hence the instruction cycle is accordingly longer. Considering that the GPU is designed for a large throughput and massively parallel computation, the TS system should be parallelized to achieve a comparable performance to their counterparts on CPU. Therefore, the TS simulation implementation on GPU in this work becomes a unified PiT+PiS scheme.

The pure-GPU computing architecture has been utilized in [6], [7]. However, the pure-GPU Parareal algorithm requires dynamic parallelism and inefficient complex stream synchronizations. For example, the parent coarse-grid under dynamic parallelism will lock GPU resources when launching child grids, and this severely limits the scalability. It is also very slow to perform complex condition checks and loops on GPUs. To avoid degradations, a multi-stream CPU-GPU hybrid PiT+PiS scheme shown in Fig. 9 (a) is proposed, which not only utilizes more resources of a single GPU but also enables a multi-GPU architecture. The general pseudocode is attached in Appendix B.

The streams are pre-defined in the initialization stage, and the kernel launching is performed on the CPU. In Fig. 9 (a) the coarse-grid stream is labeled as G stream and fine-grid streams are labeled as F streams. For the xth coarse-grid step, its prediction will be used by $(x+1)$th fine-grid kernel function, so that the xth coarse-grid and xth fine-grid kernels can be launched at the same time. To maximize concurrency, the refinement of state variables $\boldsymbol{U}$ is integrated into the loop so it can overlap with fine-grid kernel executions. Since $\boldsymbol{U}$ is used in the current iteration to launch the $(x+1)$th fine-grid kernel function, the operation in coarse-grid must be

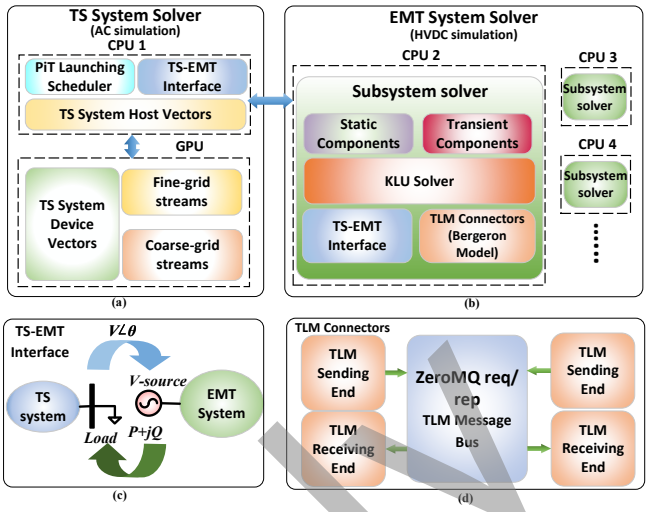

Fig. 7: TS-EMT PiT+PiS simulation program hierarchy: (a) TS system solver CPU-GPU hybrid structure based on Thrust vectors and concurrent GPU multi-streams; (b) PiS CPU multi-thread EMT system solver structure; (c) TS-EMT interface to connect the TS and EMT solvers; (d) ZeroMQ network to connect TLMs within the EMT system solver.

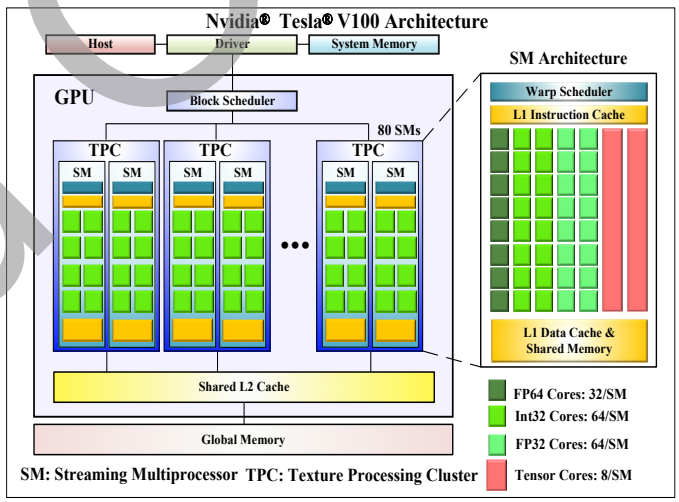

Fig. 8: General architecture of NVIDIA® Volta GPUs.

synchronized before $(x+1)$th loop iteration begins, while fine-grid F streams are fully concurrent to each other and G stream. After all coarse operations are finished, the algorithm performs a device-wide synchronization to obtain all fine-grid results which are required in the next iteration. Due to the multi-stream architecture, multi-GPU execution becomes easier since the streams can be assigned to single or multiple GPUs. The memory mitigation problem can be solved by CUDA® unified memory implicitly. The GPU multi-stream execution graph of proposed algorithm implementation is shown in Fig. 9 (b).

The memory resources allocation and management are also moved to CPU, which is much easier with the Thrust library [32]. The Thrust library is a GPU-accelerated library of C++ parallel algorithms and data structures; it provides a host-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

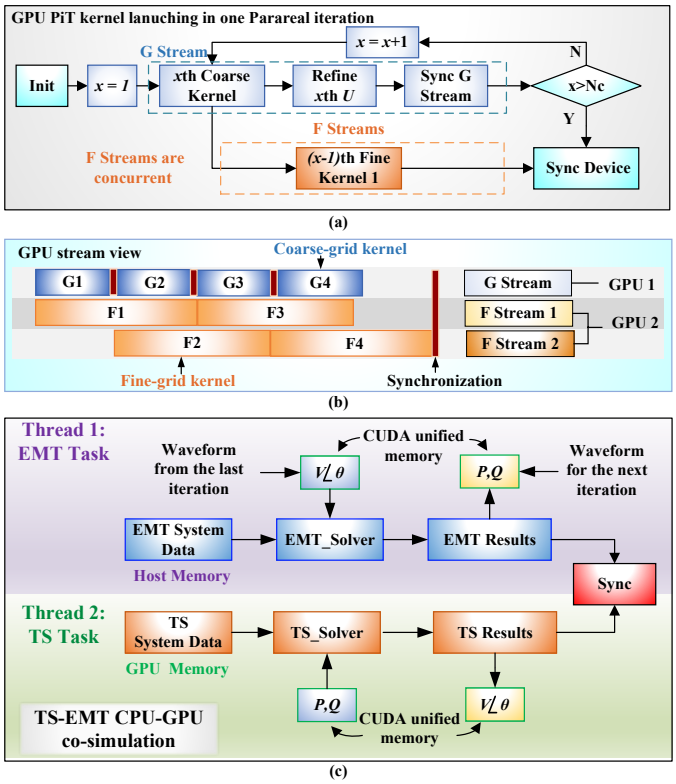

Fig. 9: Algorithm implementations of the PiT+PiS hybrid CPU-GPU TS-EMT co-simulation: (a) GPU PiT kernel launching within one Parareal iteration; (b) kernel execution graph in multi-stream/multi-GPU scenarios; (c) PiT+PiS TS-EMT co-simulation on hybrid CPU-GPU platform.

vector class and device vector class as a drop-in replacement for C++ *std::vector* class, which can easily allocate, resize, and transfer memory data between CPUs and NVIDIA® GPUs; it also provides a set of C++ *std* style functions to perform parallel operations on both CPU threads and NVIDIA® CUDA® cores.

In this work, vector classes are used to manage the memory of the PiT+PiS program, and other operations are performed by passing vector device data pointer to the hand-written CUDA® global functions, which execute the simulation step in Fig. 3. The host-vector object can be transferred to a GPU device vector object with simple *a = b;* statements; moreover, the vector classes can take an allocator template argument which allocates the vector data memory on unified/managed CUDA® memory, pinned memory, and device global memory without explicit CUDA® function calls. The hierarchy of the TS system solver is shown in Fig. 7 (a).

B. CPU-Based PiS EMT HVDC Simulation

For the EMT solver in Fig. 7 (b), static components are time-invariant power system components such as resistors, transient components are time-variant components such as capacitors, inductors, power sources, and MMCs. Due to the asynchronous nature of CPU-GPU hybrid execution and the multi-core parallel computing capability of the CPU, threads running EMT simulation can be concurrent to the TS problem solution on GPU. The granularity of EMT parallel computing is designed to be system-level, which means one thread is responsible for one or more HVDC systems. This can be achieved by OpenMP® task or simple parallel for loop construct. As shown in Fig. 7 (d), the decoupling and connections between EMT systems are based on the propagation delay of traveling wave transmission line models (TLMs) such as the Bergeron Line Model. All TLMs have a sending end and a receiving end, which are connected via a message bus implemented by ZeroMQ *req/rep* mode; the sending ends are clients to send data requests to the receiving ends, while the receiving ends are servers to accept requests and replies with their data. The *rep/req* mode of ZeroMQ is synchronous and thread-safe.

C. AC/DC Co-Simulation Interface

The AC/DC co-simulation method is based on [8] with modifications for PiT purposes, which is shown in Fig. 7 (c) and Fig. 9 (c). The general idea is that the EMT system is represented as a power source in the TS solution, while the RMS values of the bus voltages in the AC system are transformed into a time-domain instantaneous three-phase voltage source in the EMT simulation. The simulation time-step of EMT simulation is around 50μs while the time-step for TS is 100μs-10ms. Since the TS system only produces voltages and power flows in the frequency domain, the data exchange frequency can be larger than the EMT time-step. However, when using Parareal, the communication between TS and EMT system bring new challenges. The TS PiT simulation consisting of coarse-grid and fine-grid requires the information from the EMT side. If the time-window is small enough such as 1ms or 10ms, the data can be exchanged per window without any iterations. If the time-window is large, it requires restarting the EMT simulation during each typical PiT iteration. The CPU-GPU asynchronous computing architecture enables the EMT simulation to be executed on the CPU while the GPU is solving the large-scale TS problem, so the EMT simulation can be considered as acquired by free: virtually no additional executing time adds to the original TS solving process.

To avoid frequent data copies and synchronizations, the waveforms of interfacing variables are exchanged per-time window; each waveform contains sampled values at the interval of coarse-grid's time-step, which is usually 1-10ms. This method can be considered a combination of Parareal and the waveform relaxation method on AC/DC TS-EMT co-simulation. The two systems exchange the information at each TS Parareal time window as shown in Fig. 9. For most steady-state cases, the voltages, and power flow change little so the iteration can converge quickly. For sharp changes such as short circuits faults, the performance only degrades within several time-windows, but the accuracy can be guaranteed since Parareal can fall back to the sequential execution eventually.

The interface data are stored in CUDA® unified memory vectors [33]. The unified memory has a single virtual memory address for CPUs and GPUs. The data mitigation can be triggered by page faults and it is natively supported by the page mitigation engine inside NVIDIA® GPUs later than

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

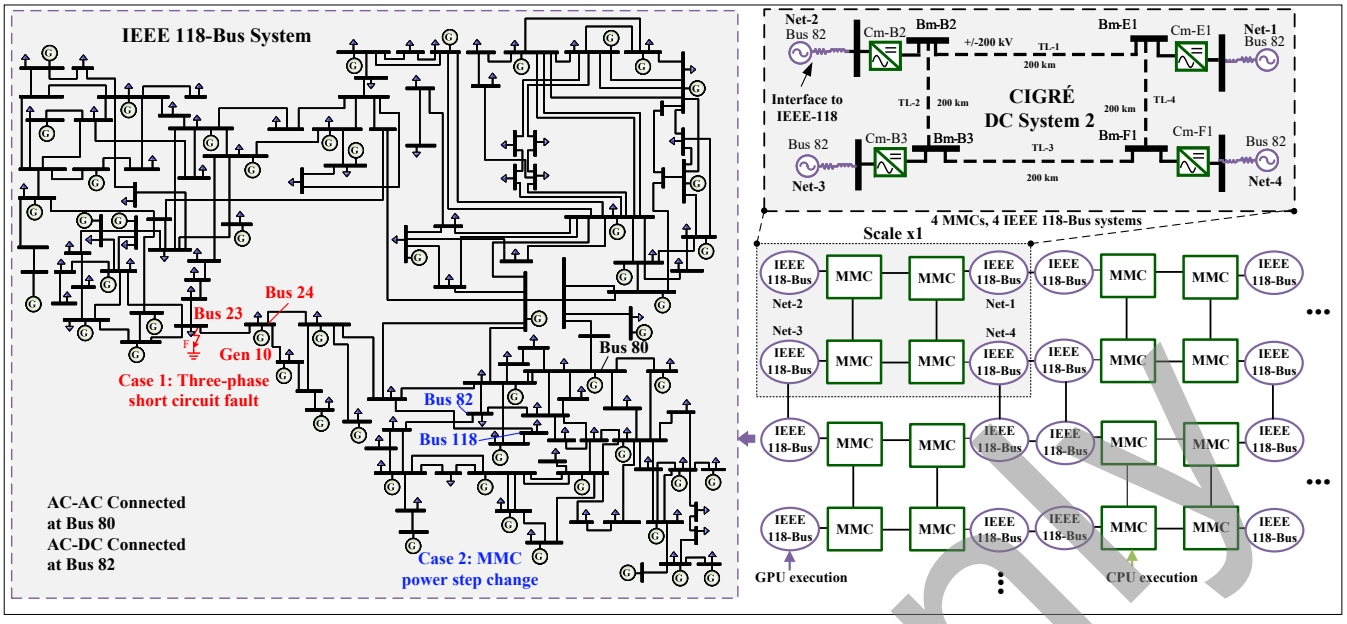

Fig. 10: Synthetic system for case studies and performance evaluations.

Maxwell® architecture. The memory can be accessed by CPU and GPUs simultaneously which is suitable for sharing data between multiple GPUs. It also enables asynchronous memory copy with *cudaMemPrefetchAsync* function [34], so that the pinned memory is not required for this task. It saves a lot of complicated memory management works for CPU-GPU and multi-GPU communications and the code can be written as the same as normal multi-thread programs on CPU.

IV. Dynamic Results and Performance Evaluation

The performance is evaluated based on the test cases shown in Fig. 10. The four IEEE 118-Bus systems and a modified CIGRÉ DCS 2 MTDC system form up the Scale x1 base test system, which is used for producing results for study Case 1 and Case 2. Then the AC/DC system is expanded vertically and horizontally as shown in Fig. 10 (a) to evaluate the scalability and computational efficiency of the hybrid CPU-GPU PiS+PiT simulation method. The 4 AC power grids in the Scale x1 system are labeled as Net-1, Net-2, Net-3, and Net-4 respectively in Fig. 10.

The Scale 2x-12x systems are only used for performance evaluation purposes. Bus 82 is chosen for the connections between HVDC grids and AC systems. For AC-AC connections, the Bus 80 of AC systems are connected. The Scale x1 system contains $4 \times 118 = 472$ TS nodes and 216 generators with four 201-level three-phase EMT modeled MMCs. The TS fine-grid simulation time-step is 100μs and coarse-grid time-step is 10ms; the EMT simulation time-step is 10μs.

A. Verifications of PiT and AC/DC Simulation

Case 1: A short circuit with a duration of 200ms happens at Bus 23. The fault resistance is 1Ω; the generator 10 is chosen as a reference for rotor angle. The main focus of this case study is to verify the results compared to the commercial

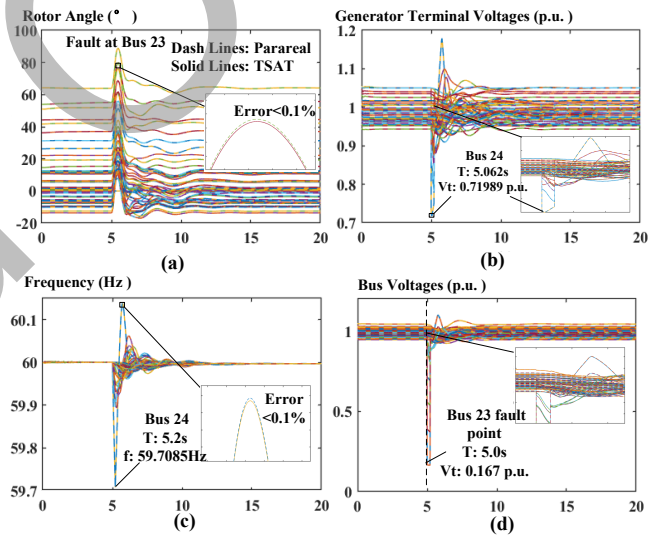

Fig. 11: Waveform of Case 1 test system: (a) generator rotor angles; (b) terminal voltages of all generators; (c) frequencies of all generators.

DSATools™ TSAT. The results shown in Fig. 11 of the Parareal algorithm meet the TSAT results very well and the relative error is smaller than 1%. After the fault, the generator transients last for two seconds then return to the normal at 10-12s since the fault is cleared. Fig. 11 (a)-(c) shows the generator rotor angles, terminal voltages, and frequencies of generators, respectively. Bus 24, which is the closest generator bus to the fault location, has the largest voltage (-0.3 p.u.) and frequency (-0.3Hz) deviations. Fig. 11 (d) shows the voltage of non-generator buses, where Bus 23 has the bus lowest voltage but not zero, which is because the fault resistance is 1Ω, not 0.

Case 2: it presents the results of PiT+PiS AC/DC co-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPWRS.2022.3153450, IEEE Transactions on Power Systems

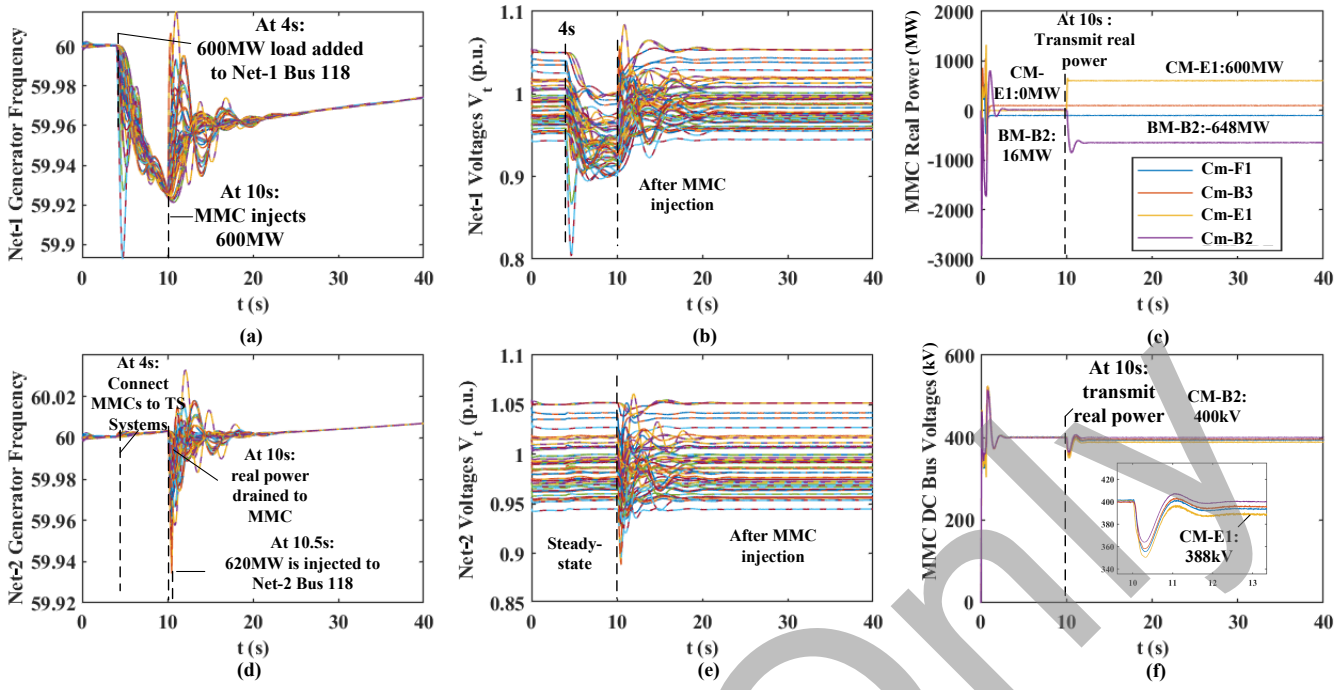

Fig. 12: Waveforms of Study Case 2: (a) Frequencies of the generators in Net-1; (b) Terminal Voltages of the generators in Net-1; (c) Real power of 4 MMCs; (d) Frequencies of the generators in Net-2; (e) Terminal Voltages of the generators in Net-2; (f) DC Voltages of four MMC terminals.

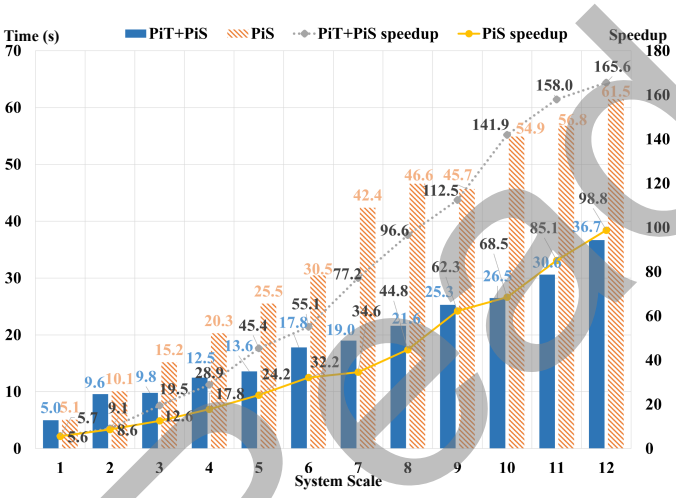

Fig. 13: Performance comparison of hybrid PiT+PiS and GPU PiS under various system scales.

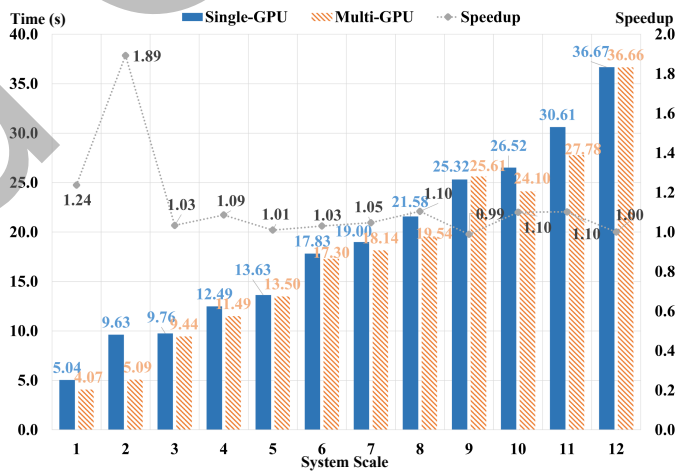

Fig. 14: Performance of the multi-GPU implementations.

simulation of an overload and recovery scenario. The MTDC system is used to support AC power systems when an overload occurs in Net-1; the Cm-B2 is working under the DC voltage control model to maintain constant DC voltages and the other MMCs are working under power control mode. This scenarios has three stages: (1) a 600MW load is added to the Bus 118 at 4s in Net-1, which causes drops in voltages and frequencies as shown in Fig. 12 (a), (b) respectively; (2) at 10s the MMC Cm-E1 is ordered to drain 600MW from HVDC buses as shown in Fig. 12 (c), thus the voltages and frequencies of Net-1 start to recover. On the other side, Cm-B2 provides the real power of 648MW to maintain DC voltages, and it has to drain real power from Net-2 so that Net-2's voltages and frequencies start to decline as shown in Fig. 12 (d), (e) respectively; (3) a 620MW real power is injected at 10.5s to Bus 118 in Net-2 so that Net-2 can maintain the stability.

B. Performance Comparison

Fig. 13 shows the performance comparison between CPU serial AC/DC co-simulation, CPU-GPU PiT+PiS AC/DC co-simulation, and the GPU PiS co-simulation with a single NVIDIA® Tesla® V100. The execution time of the CPU serial program for large-scale cases is too long so only the speedup against the serial program is presented in the plot. The speedup and execution time increases almost linearly for both PiT+PiS

and the traditional PiS parallel computing method. When the system scale is 12x, GPU PiS only achieved 98.7x speedup compared to the sequential program; meanwhile, the GPU PiT+PiS method achieved 165.6x speedup which is 1.67x faster than GPU PiS.

Fig. 14 shows the performance of PiT+PiS method with 2x NVIDIA® Tesla® V100 GPU. When the system scale is 1x and 2x, the speedup is obvious, especially when the system scale is 2x, which has 8 IEEE-118 Bus systems, the parallel-efficiency is near 100%. However, when the system scale is larger, the speedup declines to near 1.0x which indicates the GPU concurrency stops increasing under the multi-GPU situation, despite the single GPU implementation having linear speedup growth. It is due to the size of GPU kernels since the streams in Fig. 9 are not guaranteed to be concurrent. The large-size kernel amplified the load imbalance between GPU-1 and GPU-2. Because the serial coarse-grid prediction is critical to performance and it should not be delayed or disrupted, all fine-grid kernels were launched to GPU-2 while GPU-1 only handles coarse-grid tasks. As the problem size grows, coarse-grid workloads become much smaller than fine-grid workloads, so the multi-GPU results become closer to single GPU execution. The more advanced solution is to launch some of the fine-grid kernels to GPU-1 when it is idle so that the computationally intensive fine-grid tasks can make use of multiple GPUs.

V. Conclusion

A hybrid CPU-GPU parallel-in-time-and-space transient stability simulation method is proposed based on the Parareal algorithm. The Parareal algorithm is implemented on GPU along with the traditional PiS algorithm to achieve maximum parallelism. The CPU-GPU design performs PiT scheduling and launches GPU kernel functions to streams on the CPU, which brings better scalability and extensibility to GPU-only design. Meanwhile, the CPU can perform the EMT simulation asynchronously when the GPU is running the transient stability simulation, which can be perfectly integrated with the proposed AC/DC co-simulation scheme and bring better performance and parallel efficiency. The study case shows good results both in accuracy and computational performance. The speedup for the PiT+PiS method to the PiS method is around 2x and can achieve 165.6x compared to sequential CPU programs for a large-scale system. The method can utilize multiple GPUs and can achieve near-maximum parallel efficiency with a system scale of 2x. Further investigations and benchmarks are planned to find the bottleneck and optimize the PiT+PiS algorithm on multi-GPUs. The proposed hybrid PiT+PiS method shows good potential for the solution of large-scale AC/DC power system transient stability simulation problems.

Appendix A

Test Environment: Parallel computing node of Compute Canada Cedar cluster; Two Intel® Xeon® Silver 4216 Cascade Lake processors (16 Cores, Base frequency: 2.1GHz, max turbo frequency 3.2GHz, cache L3: 22MB, cache L2: 16MB). Software configuration: Operating System: CentOS 7.7, CUDA® 11.0, Linux 3.10 kernel; Compiler: GCC/G++ 9.3, OpenMP 4.5.

Appendix B

Top level code of CPU-GPU asynchronous program

```
len = 10000, coarse_len = 10, fine_len = 100
tol = 1e-4
# objects for coarse and fine-grid workers
coarse_network = create_coarse_network()
fine_networks = create_fine_networks()
# coarse_network is initalized with larger dt

# solution vectors for Parareal
Gk, Gk_1, Fk, Uk = create_parareal_vectors()


# TS-EMT data exchange vector, unified memory
PQ, Vtheta = create_TSEMT_vectors()
iter = 0
maxit = 10
converged = False
Gstream, Fstreams = create_streams()
for i in range(0, len):
  while not converged and iter < maxit:
    # Thread 1: # EMT simulation Task
    Prepare_emt_sim()
    EMT_solve()
    PrefetchDataToGPU(PQ)
    # Thread 2: # TS simulation Task
    # set intial values
    Gk[0] = Uk[0]
    coarse_network = Uk[0]
    fine_networks[0] = Uk[0]
    # 10 intervals has 11 coarse points
    for x in range(1, coarse_len+1):
      # PiS GPU functions are called in
      # the solve function
      TS_coarse_solve(
          coarse_network, Gk[x]
          PQ, Vtheta, Gstream)

      if iter == 0:
        # (a) coarse prediction
        # copy functions are aync kernel functions
        # the stream is assigned at runtime
        copy_from_to(Gk[x], Uk[x], Gstream)
      else:
        # (c) solution refinement
        err = Gk[x] - Gk_1[x]
        converged = converged & check_error(err, tol)
        Uk[x] = Fk[x * fine_len] + err
        copy_from_to(Uk[x], coarse_network, Gstream)

      copy_from_to(Uk[x], finenet[x], Gstream)
      # (b) Fine-grid parallel operation
      for xt in range(0, fine_len):
        TS_fine_solve(
            fine_networks[x-1],
            PQ, Vtheta, Fstreams[x-1])
        Fk_idx = x*fine_len+xt+1
        copy_from_to(
            fine_networks[x-1],
            Fk[Fk_idx], Fstreams[x-1])
      # ensure results before next serial operation
      cudaStreamSynchronize(Gstream)

    copy_from_to(Gk, Gk_1, Gstream)
    # device only synchronize once in 1 iteration
    # since fine-grids kernels are concurrent
    cudaDeviceSynchronize()
    PrefetchDataToCPU(Vtheta)
# (d) the final solution of a window is there
```

References

[1] "Changji-Guquan UHVDC transmission project," 2015. [Online]. Available: http://www.nsenergybusiness.com/projects/changji-guquan-uhvdc-transmission-project/, [Accessed: 13-Feb-2022].

[2] "Dolwin 5 HVDC project." [Online]. Available: https://www.hitachiabb-powergrids.com/ca/en/references/hvdc/dolwin-5, [Accessed: 13-Feb-2022].

[3] "Critical grid infrastructure to connect the west." [Online]. Available: http://www.transwestexpress.net/, [Accessed: 13-Feb-2022].

[4] M. La Scala, M. Brucoli, F. Torelli, and M. Trovato, "A gauss-jacobi-block-newton method for parallel transient stability analysis (of power systems)," *IEEE Trans. Power Syst.*, vol. 5, no. 4, pp. 1168–1177, Nov 1990.

[5] S. Xia, S. Bu, J. Hu, B. Hong, Z. Guo, and D. Zhang, "Efficient transient stability analysis of electrical power system based on a spatially paralleled hybrid approach," *IEEE Trans. Ind. Informat.*, vol. 15, no. 3, pp. 1460–1473, 2019.

[6] 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, 2010.

[7] 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, 2012.

[8] N. Lin, S. Cao, and V. Dinavahi, "Comprehensive modeling of large photovoltaic systems for heterogeneous parallel transient simulation of integrated ac/dc grid," *IEEE Trans. Energy Convers.*, vol. 35, no. 2, pp. 917–927, June 2020.

[9] ——, "Adaptive heterogeneous transient analysis of wind farm integrated comprehensive ac/dc grids," *IEEE Trans. Energy Convers.*, vol. 36, no. 3, pp. 2370–2379, 2021.

[10] V. Dinavahi and N. Lin, *Parallel Dynamic and Transient Simulation of Large-scale Power Systems*, Springer Nature Switzerland AG, Cham, Switzerland, 2022.

[11] T. Carraro, M. Geiger, S. Rorkel, and R. Rannacher, *Multiple Shooting and Time Domain Decomposition Methods*. Springer, New York, 2015.

[12] M. La Scala, R. Sbrizzai, and F. Torelli, "A pipelined-in-time parallel algorithm for transient stability analysis (power systems)," *IEEE Trans. Power Syst.*, vol. 6, no. 2, pp. 715–722, May 1991.

[13] M. La Scala, G. Sblendorio, and R. Sbrizzai, "Parallel-in-time implementation of transient stability simulations on a transputer network," *IEEE Trans. Power Syst.*, vol. 9, no. 2, pp. 1117–1125, May 1994.

[14] Y. Takahashi, K. Fujiwara, T. Iwashita, and H. Nakashima, "Parallel finite-element method based on space–time domain decomposition for magnetic field analysis of electric machines," *IEEE Trans. Magn.*, vol. 55, no. 6, pp. 1–4, June 2019.

[15] S. Schöps, I. Niyonzima, and M. Clemens, "Parallel-in-time simulation of eddy current problems using parareal," *IEEE Trans. Magn.*, vol. 54, no. 3, pp. 1–4, Mar. 2018.

[16] B. Park, K. Sun, A. Dimitrovski, Y. Liu, and S. Simunovic, "Examination of semi-analytical solution methods in the coarse operator of parareal algorithm for power system simulation," *IEEE Trans. Power Syst.*, pp. 1–1, 2021.

[17] G. Gurrala, A. Dimitrovski, S. Pannala, S. Simunovic, and M. Starke, "Parareal in time for fast power system dynamic simulations," *IEEE Trans. Power Syst.*, vol. 31, no. 3, pp. 1820–1830, May 2016.

[18] D. Osipov, N. Duan, S. Allu, S. Simunovic, A. Dimitrovski, and K. Sun, "Distributed parareal in time with adaptive coarse solver for large scale power system simulations," *2019 IEEE Power Energy Society General Meeting (PESGM)*, pp. 1–5, 2019.

[19] "IEEE Guide for Synchronous Generator Modeling Practices and Parameter Verification with Applications in Power System Stability Analyses", in IEEE Std 1110-2019 (Revision of IEEE Std 1110-2002) , vol., no., pp.1-92, 2 March 2020.

[20] P. Kundur, N. J. Balu, and M. G. Lauby, *Power system stability and control*. McGraw-Hill New York, 1994.

[21] "IEEE recommended practice for excitation system models for power system stability studies," *IEEE Std 421.5-2005 (Revision of IEEE Std 421.5-1992)*, pp. 1–93, 2006.

[22] V. Vittal, J. D. McCalley, P. M. Anderson, and A. Fouad, *Power system control and stability*. John Wiley & Sons, Hoboken, New Jersey, 2019.

[23] W. Janischewskyj and P. Kundur, "Simulation of the non-linear dynamic response of interconnected synchronous machines part i-machine modelling and machine-network interconnection equations," *IEEE Trans. Power App. Syst.*, vol. PAS-91, no. 5, pp. 2064–2069, 1972.

[24] B. Stott, "Power system dynamic response calculations," *Proceedings of the IEEE*, vol. 67, no. 2, pp. 219–241, 1979.

[25] V. Dinavahi and N. Lin, *Real-time electromagnetic transient simulation of AC-DC networks*, Wiley-IEEE Press, Hoboken, New Jersey, 2021.

[26] Q. Tu and Z. Xu, "Impact of sampling frequency on harmonic distortion for modular multilevel converter," *IEEE Trans. Power Del.*, vol. 26, no. 1, pp. 298–306, 2011.

[27] T. A. Davis and E. P. Natarajan, "Algorithm 907: KLU, A direct sparse solver for circuit simulation problems," *ACM Trans. Math. Softw.*, vol. 37, no. 3, pp. 36:1–36:17, 2010.

[28] M. J. Gander and S. Vandewalle, "Analysis of the parareal time-parallel time-integration method," *SIAM Journal on Scientific Computing*, vol. 29, no. 2, pp. 556–578, 2007.

[29] M. J. Gander and E. Hairer, "Nonlinear convergence analysis for the parareal algorithm," in *Domain decomposition methods in science and engineering XVII*. Springer, New York, 2008, pp. 45–56.

[30] J. L. Gustafson, *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011, pp. 53–60.

[31] M. Minion, "A hybrid parareal spectral deferred corrections method," *Commun. Appl. Math. Comput. Sci.*, vol. 5, no. 2, pp. 265 – 301, 2010.

[32] "Thrust: Code at the speed of light." [Online]. Available: https://github.com/NVIDIA/thrust, [Accessed: 13-Feb-2022].

[33] "Unified memory for cuda beginners," June 2017. [Online]. Available: https://developer.nvidia.com/blog/unified-memory-cuda-beginners, [Accessed: 13-Feb-2022].

[34] "Maximizing unified memory performance in cuda," Nov 2017. [Online]. Available: https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda, [Accessed: 13-Feb-2022].

Tianshi Cheng (S'19) received the B.Eng. degree in electrical engineering and automation from Southeast University, China, in 2017. From 2017 to 2018, he was a substation automation engineer of NARI Group Corporation (State Grid Electric Power Research Institute), China. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Alberta, Canada. His research interests include electromagnetic transient simulation, transient stability analysis, real-time simulation, parallel processing, microgrid and power electronics.

Ning Lin (S'17-M'19) received the B.Sc. and M.Sc. degrees in Electrical Engineering from Zhejiang University, China, in 2008 and 2011, respectively, and the Ph.D. degree in Electrical and Computer Engineering from the University of Alberta, Edmonton, AB, Canada, in 2018. From 2011 to 2014, he was an engineer on a flexible AC transmission system (FACTS) and high-voltage direct current (HVDC) transmission. His research interests include electromagnetic transient simulation, transient stability analysis, real-time simulation, device-level modeling, integrated AC/DC grids, massively parallel processing, heterogeneous high-performance computing of power systems, and power electronics.

Venkata Dinavahi (Fellow, IEEE) received the B.Eng. degree in electrical engineering from Visvesvaraya National Institute of Technology (VNIT), Nagpur, India, in 1993, the M.Tech. degree in electrical engineering from the Indian Institute of Technology (IIT) Kanpur, India, in 1996, and the Ph.D. degree in electrical and computer engineering from the University of Toronto, Ontario, Canada, in 2000. He is currently a Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada. He is a Fellow of the Engineering Institute of Canada. His research interests include real-time simulation of power systems and power electronic systems, electromagnetic transients, devicelevel modeling, large-scale systems, and parallel and distributed computing.