Scientific Research Publishing

Journal of Computer and Communications, 2020, 8, 39-63
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Graphical Processing Unit Based Time-Parallel Numerical Method for Ordinary Differential Equations

Sumathi Lakshmiranganatha, Suresh S. Muknahallipatna*

Department of Electrical and Computer Engineering, University of Wyoming, Laramie, WY, USA
Email: slakshmi@uwyo.edu, *sureshm@uwyo.edu

How to cite this paper: Lakshniranganatha, A. and Muknahallipatna, S.S. (2020) Graphical Processing Unit Based Time-Parallel Numerical Method for Ordinary Differential Equations. *Journal of Computer and Communications*, **8**, 39-63.
https://doi.org/10.4236/jcc.2020.82004

Received: December 30, 2019
Accepted: February 25, 2020
Published: February 28, 2020

Abstract

On-line transient stability analysis of a power grid is crucial in determining whether the power grid will traverse to a steady state stable operating point after a disturbance. The transient stability analysis involves computing the solutions of the algebraic equations modeling the grid network and the ordinary differential equations modeling the dynamics of the electrical components like synchronous generators, exciters, governors, etc., of the grid in near real-time. In this research, we investigate the use of time-parallel approach in particular the Parareal algorithm implementation on Graphical Processing Unit using Compute Unified Device Architecture to compute solutions of ordinary differential equations. The numerical solution accuracy and computation time of the Parareal algorithm executing on the GPU are demonstrated on the single machine infinite bus test system. Two types of dynamic model of the single synchronous generator namely the classical and detailed models are studied. The numerical solutions of the ordinary differential equations computed by the Parareal algorithm are compared to that computed using the modified Euler's method demonstrating the accuracy of the Parareal algorithm executing on GPU. Simulations are performed with varying numerical integration time steps, and the suitability of Parareal algorithm in computing near real-time solutions of ordinary different equations is presented. A speedup of 25× and 31× is achieved with the Parareal algorithm for classical and detailed dynamic models of the synchronous generator respectively compared to the sequential modified Euler's method. The weak scaling efficiency of the Parareal algorithm when required to solve a large number of ordinary differential equations at each time step due to the increase in sequential computations and associated memory transfer latency between the CPU and GPU is discussed.

Keywords

Time-Parallel, Differential Equation, Numerical Integration, Graphic Processing Unit

1. Introduction

The time-domain simulation technique is widely used by the power industry to describe a power grid transient behavior accurately. The high level of accuracy achieved using the time-domain simulation technique is due to the use of detailed mathematical models of controls, nonlinearity, saturation, and protection systems. Power system stability studies or analysis typically involve computing the system response to a sequence of large disturbances, such as generator outage or network short circuit, followed by a switching operation as part of protective measures. The system response computation involves a direct simulation in the time-domain of duration varying between 1 s and 20 min., or more. The system response or stability at different stages of time-domain simulation is affected by different components of the grid dictated by the level of mathematical modeling of the individual components used.

The power system stability studies using the time-domain simulation technique are performed using two levels of mathematical models of the grid components, namely the short-term and long-term models. The short-term models represent the rapidly responding system electrical components of the power grid like generators, exciters, governors, turbines, etc., while the long-term models represent the slow-oscillatory system power balance (variable load). Using the short-term models to perform power system stability studies addressing the post-disturbance times of up to 5 - 10 secs is classified as "Transient Stability" Analysis (TSA) whereas using the long-term models to stability studies are associated with frequency and voltage stability. The focus of the research work presented in this paper is on the Transient Stability Analysis of the power system.

The TSA involves computing the step-by-step solution of thousands of non-linear systems of coupled differential-algebraic equations (DAEs) representing the dynamic components and the network interconnect of the dynamic components of the power system. The TSA, in particular, is concerned with the simulation of faults and contingencies, which can produce instability of the power system. The focus is to simulate a number of possible contingencies in a short-time horizon to evaluate possible instability conditions and develop appropriate corrective actions. The preventive simulation and corresponding corrective action are repeated for tens or hundreds of cases, until a system or utility operator by an online evaluation of the power system state, detects unsafe operating conditions. These exhaustive and computationally intensive simulations are performed to provide an operator with appropriate corrective action that can be triggered when a contingency occurs in real-time. However, the online TSA, in particular,

is a computationally challenging problem, requiring 10 - 15 minutes to perform preventive simulation of a power system (depends on the size of the power system) for a set of fault conditions and outages [1]. In 1990s, a number of researchers explored the use of traditional time-domain simulation and innovative computer architectures like parallel/vector processing and distributed computing [2] [3] and [4] to achieve online TSA and demonstrated the limited amount of parallelism that could be exploited.

The parallelization techniques that can be applied to perform a transient stability analysis of a power system can be broken into spatial domain decomposition, numerical method, and temporal domain decomposition or time-parallel parallelism approaches. The spatial decomposition approach [5] [6] involves partitioning the system DAEs and distributing the computations over various processors. The parallelism across numerical method approach is to exploit the parallelism in the numerical scheme used to solve the DAEs. The approach is to use waveform relaxation, VDHN-Maclaurin numerical schemes [7] [8] instead of the traditional trapezoidal, Runge-Kutta, and Adam-Bashford methods. The temporal decomposition approach involves dividing the integration interval into blocks and solving the blocks in parallel.

The first two parallelization techniques have been researched and available in some commercial packages like PSS-E, PowerWorld, and OPAL-RT [9]. The research focus has been on using task level and distributed computing across multiple contingency analysis but not on speeding up the computations in a single case. The use of the time-parallel parallelism approach in other fields was first considered by Nievergelt [10], and he presented a method for parallelizing the numerical integration of an ordinary differential equation. In 1979, Alvarado proposed the use of time-parallel with trapezoidal algorithm [11] for transient problems. However, the implementation of the proposed approach could not achieve significant computational speedup due to the coupling between adjacent time steps resulting in sequential execution. Later, pipelining [12] and relaxation-based techniques [13] were proposed to implement time-parallel in single case. The speedup gain of the pipelined-in-time was limited due to sequential convergence, while the gain with the relaxation-based techniques was limited due to slow convergence. Many time-parallel approaches [14] are applicable to certain classes of problems only and dependent on specific computer architectures.

In this paper, we investigate the use of the time-parallel approach and in particular the Parareal algorithm (PRA) implementation on the Graphical Processor Unit (GPU) using the Compute Unified Device Architecture (CUDA) for solving ODEs representing the electrical components of the power system. The investigation in [10] on the use of parallel methods for integrating ODEs was first attempted in 1964, and later in the year 2002, solving ODEs using the predictor-corrector form of the PRA [15] [16] was investigated. For a decade from 2002, a number of researchers have worked on establishing the stability and convergence properties [17] [18], scaling of the algorithm performance with the

number of computing units [19] [20], distribution of workload [21] [22], and application to solve a large class of traditional problems involving the solutions of non-linear parabolic equations [23], nonlinear differential equations in the financial world [24], equations of molecular dynamics [16], quantum chemistry [25], partial differential equations in optimal control, etc., to mention a few. Recently, the use of time-parallel algortihms to reduce the computational time of power system dynamics simulation has been investigated by a few researchers.

The implementation of PRA has been investigated in [26] for the detailed models of power systems for TSA. The authors have investigated different numerical integration methods for coarse solvers to analyze the stability and convergence property of the algorithm. They achieved consistent results with the midpoint trapezoidal predictor-corrector method for coarse solver and RK4 method for the fine solver of PRA. The performance of PRA is compared with the Newton-based time-parallel methods using a single machine infinite bus system (SMIB). The computational speedup of 10× and 40× is achieved with 150 and 250 processors for two cases of the SMIB with varying number of steps. The authors have evaluated the stability and convergence properties of the algorithm for two large scale power systems. The authors in [27] reduce the computation time by using a simplified generator model for the coarse solver stage in the PRA implementation. A 13% improvement in the execution time was observed between the simplified generator model and the detailed model for the Polish power system. The overall algorithm speedup achieved was ~10× for the Polish system. The communication overhead between the processors is neglected in the speedup calculation.

The research in [28] shows that embedding the spatial decomposition into PRA has better performance than using either one of them individually. Each system is spatially divided into two sub-areas and is solved in parallel. Each sub-area is solved using PRA to achieve time parallelism and reduction in execution time. The research demonstrated a ~33% improvement in the parallel fine steps execution time between the hybrid method proposed and only PRA. A new approach was proposed to perform TSA in [29], which is a spatially parallel hybrid approach combining the high-order Taylor series and the block bordered diagonal form (BBDF) to reduce the computational burden of TSA. They propose using only the higher-order derivatives of generator voltages and currents along with large integration time steps to perform TSA. Three systems are used for testing the proposed approach. The speedup they demonstrate by using only higher derivatives of the generators; the equations are decoupled from the power network equation, leading to spatial and time-domain parallel decomposition hybrid approach. A factor of ~2× is achieved with the proposed method compared to other commercially available parallel integrators.

In the above investigations of the TSA using time-parallel approach, all of the implementations have been on a single node or multi-node clusters based on the Intel processors using MATLAB programming language for the algorithm im-

plementation. The speedup achieved does not include the communication overhead between multiple cores on a single node and between the nodes in a cluster. Furthermore, the research in all of the above investigations is on evaluating the suitability of PRA for transient stability analysis. In this paper, we investigate the performance of PRA to solve ODEs using heterogeneous computing architecture, namely the use of massive parallel cores on the graphical processing units (GPU) with compute unified device architecture (CUDA) [30]. The focus of the investigation is to develop a reliable implementation of PRA on CUDA architecture to solve ODEs in temporal decomposition to reduce computational time and which can be applied to achieve real-time or faster than real-time TSA with a large number of GPUs.

This paper is organized as follows: In section 2, the PRA with the Predictor Correction approach is discussed. Section 3 gives a brief overview of power system modeling, in particular, the classical and fourth-order models of synchronous generator. In section 4, we present the implementation of Parareal algorithm on NVIDIA GPUs, and in section 5, the simulation results and analysis are presented. Section 6 presents the conclusion and future work.

2. Parareal Algorithm

The origin of PRA can be traced back to spatial domain decomposition technique. The PRA involves dividing the entire simulation time T into small sub-intervals and solving these subintervals in parallel. Initial conditions are required to solve the small sub-intervals in parallel. The initial conditions are provided by a fast but less computationally expensive sequential numerical integrator. The small sub-intervals are then solved in parallel to get more accurate solution of an ODE.

Consider a general nonlinear ODE with given initial condition, $u(0)=u^0$ as shown in Equation (1)

$$\dot{u} = f(u,t), t \in [0,T] \tag{1}$$

The entire simulation time t is decomposed into N sub-intervals as $T_0 < T_1 < \cdots < T_N$ with the step size of $\Delta T = T_n - T_{n-1}, \forall 1 \le n < N$.

Two numerical operators, namely, coarse and fine propagators are defined in PRA. The coarse propagator denoted as $G_{\Delta T}$, operates using the initial condition $u(T_{n-1}) = U_{n-1}$ is used to compute the approximate solution of Equation (1) with time step ΔT at time T_n as shown in **Figure 1**. The approximate solution computed by the coarse propagator is denoted as $\widehat{U_n}$. The solution obtained using coarse propagator is less accurate but is computationally inexpensive. The coarse propagator is mathematically represented in Equation (2).

$$\widetilde{U_n} = G_{\Delta T}\left(T_{n-1}, \widetilde{U_{n-1}}, \Delta T\right), \widetilde{U_0} = u^0 \tag{2}$$

The fine propagator denoted as $F_{\delta t}$, also will use the initial condition $u(T_{n-1}) = U_{n-1}$ to compute approximate solution of Equation (1) with smaller time step $\delta t \ll \Delta T$ at time T_n as shown in **Figure 2**.

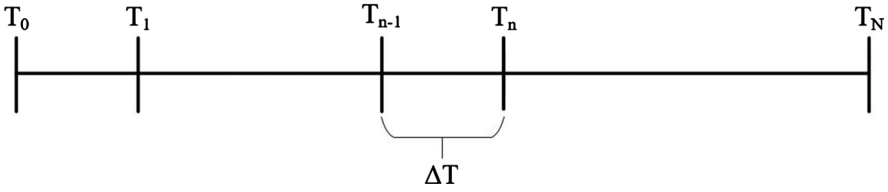


Figure 1. Decomposition of time into smaller sub-intervals.

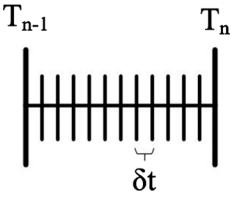


Figure 2. Fine propagator computation using time step δt.

The solution computed from the fine propagator is denoted as $\widehat{U_n}$. The solution obtained from the fine propagator is more accurate compared to coarse propagator but it is computationally expensive. The fine propagator is mathematically described in Equation (3)

$$\widehat{U_n} = F_{\delta t}\left(T_{n-1}, \widehat{U_{n-1}}, \delta t\right), \widehat{U_0} = u^0 \tag{3}$$

The flowchart of the implementation of PRA is shown in **Figure 3**.

The flow chart consists of the three steps of the PRA which are discussed below:

Step 1: The initial step of PRA is the computation of the initial conditions sequentially using the coarse propagator that is used for solving the sub-intervals in parallel. The initial coarse propagator generates a fast but less accurate initial conditions using Equation (4).

$$U_n^0 = \widetilde{U_n^0} = G_{\Delta T}\left(U_{n-1}^0\right), \forall 1 \le n < N \tag{4}$$

where, the superscript "0" denotes the initial iteration.

This step is performed to initialize the PRA iterations.

Step 2: The fine propagator is used to propagate the fine solution in parallel over each sub-interval $t \in [T_{n-1}, T_n]$ as

$$\widehat{U_n^k} = F_{\delta t}\left(U_{n-1}^{k-1}\right), \forall 1 \le n < N \tag{5}$$

where, $k = 1, 2, \cdots, k_{\max}$ is the iteration number.

Step 3: Once the fine solutions are obtained using the coarse solutions as initial conditions, the PRA corrects the sequential coarse predictions using predictor-corrector method. Predictor-Corrector method is used to correct the solution difference obtained from coarse and fine propagators for the next iteration. The predictor-corrector scheme is described in Equation (6).

$$U_n^k = \widehat{U_n^k} + \widetilde{U_n^k} - \widetilde{U_n^{k-1}}, \forall 1 \le n < N \tag{6}$$

where,

$$\widetilde{U_n^k} = G_{\Delta T}\left(U_{n-1}^k\right) \tag{7}$$

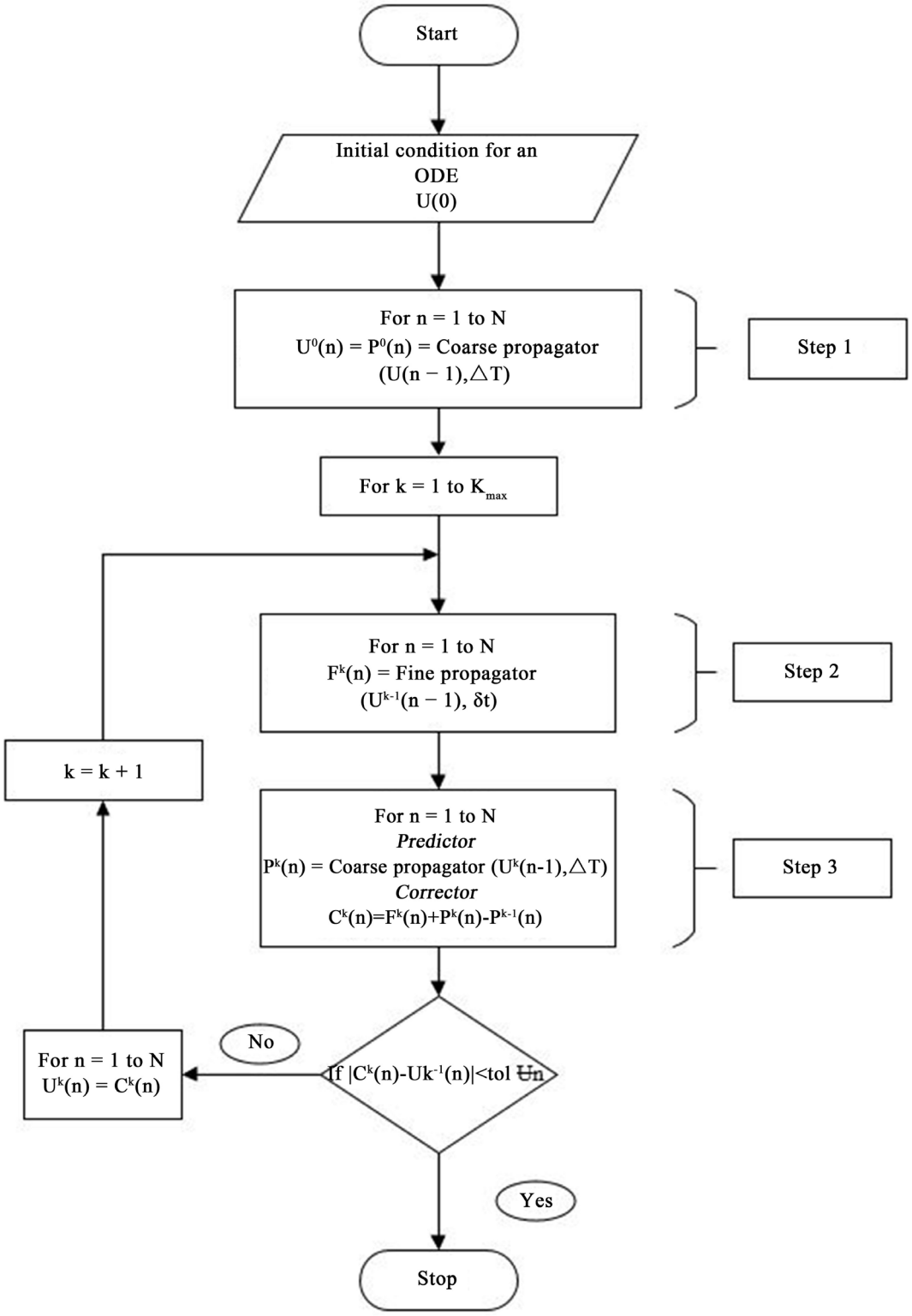


Figure 3. PRA flowchart.

The notation U represents the correct coarse solution which is used as the initial conditions for step 2 of the next iteration. At the end of 1st iteration, the coarse value at time T_1 gets corrected to the fine solution. Similarly, at the end of

the kth iteration, the coarse value at time T_k will get corrected to its respective fine solution. The steps 2 and 3 of the algorithm is iterated until the difference between the two successive coarse values meets the desired tolerance level shown in Equation (8).

$$\left|U_n^k - U_n^{k-1}\right| \le tol, \forall 1 \le n < N \tag{8}$$

The coarse solutions are generally less accurate and play an essential role in the convergence of the algorithm [14]. The choice of the time step for coarse propagator is usually bigger than that of fine propagator. However, choosing a large time step for the coarse propagator results either in a large number of iterations to converge, or not converging. Hence, the time step selection for the coarse propagator influences the number of iterations required to compute the solution.

3. Ordinary Differential Equations Representing the Power System Dynamics

Power system dynamics are modeled as a set of Differential-Algebraic equations (DAE) of the form

$$\dot{x} = f(x, y, u) \tag{9}$$

$$g(x, y) = 0 \tag{10}$$

The set of differential Equation (9) describes the behavior of all dynamic elements of a power grid like generators, exciters, governors, turbines, etc. The set of algebraic Equation (10) describes the power grid network connectivity, and all the static elements, *i.e.*, static load. The x represents the system dynamic state variables of the power grid, and is dependent on the level of models of the dynamic elements [31]. For example, at the lowest model level, x represents only the generator rotor angle and angular velocity, while at other levels it can represent various generator voltages, exciter field voltages, governor frequency control parameters. Depending on the level of modeling of the dynamic elements, the number of ODEs in the set (9) can vary from two to twenty-seven. In this work, two levels of modeling namely the model 1.0 or more commonly known as the classical model, and the model 1.1 or fourth-order generator model are considered. The solution of the ODEs during a fault present the dynamic state of the power system and the analysis is known as TSA. The variable y represents the algebraic variables such as bus voltage, bus angle, real and reactive power, etc. and angle of the power system.

3.1. Classical Model of Synchronous Generators

The classical model primarily focuses on modeling the rotor angle and angular velocity of the generator when subjected to a disturbance. When the power system is subjected to a disturbance, the rotor of the synchronous generator will accelerate or decelerate with respect to rotating magnetic field which causes rela-

tive motion. The relative motion of electromechanical oscillations of the synchronous generator is represented as "Swing equation" [32]. Equation (11) is the classical mathematical model representation of the swing equation as a second order ODE.

$$\frac{H}{\pi f_o}\frac{\mathrm{d}^2\delta}{\mathrm{d}t^2} = P_m - P_e = P_a \tag{11}$$

where,

H is the inertia constant (MJ/MVA).

P_m is the mechanical input power.

P_e is the electrical output power, where $P_e = \frac{E'V}{X}\sin\delta$.

E' is the internal EMF of the generator.

V is the terminal voltage.

$$X = X_d' + X_t + X_l$$

X_d' is the d axis transient reactance.

X_t is the transformer reactance.

X_l is the line reactance.

the difference, $P_m - P_e$ is known as the accelerating power P_a.

f_o is the nominal frequency.

$\omega_s = 2\pi f_o$ is the rated angular speed.

δ *is* the rotor angle.

$\omega = \frac{\mathrm{d}\delta}{\mathrm{d}t}$ is the relative speed or angular velocity with respect to the synchronously revolving magnetic field (reference frame).

The variation of the two state variables δ and ω with respect to time due to a disturbance is mathematically modeled by two first order ODEs shown below:

$$\frac{\mathrm{d}\delta}{\mathrm{d}t} = \omega - \omega_s = \Delta\omega \tag{12}$$

$$\frac{\mathrm{d}\Delta\omega}{\mathrm{d}t} = \frac{\pi f_0 P_a}{H}. \tag{13}$$

3.2. Detailed Model of Synchronous Generators

The detailed model of a synchronous generator addresses the direct and quadrature axis parameters of a synchronous generator taking into account the saliency. A salient pole synchronous generator is represented with the steady state and transient reactances on both direct and quadrature axis along with corresponding voltages and currents. Four-time dependent ODEs [33], Equation (14) through (17) model mathematically the dynamic behavior of the generator. The four ODEs model is commonly referred as the fourth-order model of the synchronous generator.

$$\frac{\mathrm{d}\delta}{\mathrm{d}t} = \omega - \omega_s = \Delta\omega \tag{14}$$

$$T'_{d0}\frac{\mathrm{d}E'_q}{\mathrm{d}t} = -E'_q - \left(X_d - X'_d\right)i_d + E_{fd} \tag{15}$$

$$T'_{q0}\frac{\mathrm{d}E'_d}{\mathrm{d}t} = -E'_d + \left(X_q - X'_q\right)i_q \tag{16}$$

$$\frac{H}{\pi f_o}\frac{\mathrm{d}\omega}{\mathrm{d}t} = T_m - T_e - D\left(\omega - \omega_s\right) \tag{17}$$

where,

E'_d and E'_q are the transient voltages along direct (d) and quadrature (q) axis respectively of the generator.

i_d and i_q are the stator currents of the d and q axis respectively.

D is the damping constant.

X_d and X_q are the d and q axis synchronous reactances respectively.

X'_d and X'_q are the d and q transient reactances respectively.

T'_{d0} and T'_{q0} are the open-circuit transient time constants for d and q axes.

T_m and T_e are the mechanical and electrical torque, respectively.

The electrical torque T_e is given by the Equation (18) below,

$$T_e = E'_d i_d + E'_q i_q + \left(X'_q - X'_d\right)i_q i_d \tag{18}$$

To compute T_e using Equation (18), two algebraic equations involving the stator parameters of the generators have to be solved.

Therefore, the set of differential equations modeling the dynamics of the power system with classical model of the synchronous generators will consist of two first order ordinary differential equations, and with the detailed model will consists of four first order ordinary differential equations along with three algebraic equations. The TSA of a power system due to a disturbance will involve solving a set of first order ODEs of each generator in a power system using a suitable numerical integration method. Since the generator ODEs are typically stiff the time step used in the numerical integration method has to be small to compute an accurate solution and not encounter numerical integration instability.

The number of ODEs modeling the dynamics of a generator depending on the level of modeling can vary from two to twenty seven when all of the control devices like exciter, governor, turbine, stabilizer, etc., are included. A typical power grid having in excess of thousands of generators, the TSA involves computing the numerical integration solution of in excess of ten thousands of ODEs to determine the stability of the grid, necessitating the use of Parareal algorithm executing on GPUs.

4. Implementation

4.1. Numerical Method

The two state variables δ and ω variation in time due to a disturbance is determined by solving the Equations (12) and (13) in case of classical model and Equations (14) and (17) in case of detailed model using a suitable numerical integration method. The ODEs representing the classical and detailed models being

stiff requires the use of an explicit integration method like modified Euler's method. The modified Euler's method is used by both coarse and fine propagators. The modified Euler's method is also known as the predictor-corrector method. The modified Euler's method is a single-step method, which given the initial values for an interval (t_{n-1}, t_n), the approximate solution at t_n is obtained in two steps:

Step 1: Predictor

In this step, the approximate solution y_n^p is computed using the explicit Euler's method with time step size h described by the Equation (19).

$$y_n^p = y_{n-1} + hf\left(x_{n-1}, y_{n-1}\right) \tag{19}$$

where $hf\left(x_{n-1}, y_{n-1}\right)$ is the slope of the tangent at point $\left(x_{n-1}, y_{n-1}\right)$.

Step 2: Corrector

Using the predicted y_n^p solution from step 1, the corrected solution y_n is computed using equation 20. The correction involves calculating the average of the slopes at points $\left(x_{n-1}, y_{n-1}\right)$ and $\left(x_n, y_n^p\right)$ and adding it to the corrected solution in the previous time step.

$$y_n = y_{n-1} + \frac{h}{2}\left\{f\left(x_{n-1}, y_{n-1}\right) + f\left(x_n, y_n^p\right)\right\} \tag{20}$$

Therefore at each time step, an approximate solution is first computed and then a corrector is applied to improve the approximate solution of the state variables.

4.2. GPGPU Based Parareal Algorithm Implementation

General Purpose Computing on GPU (GPGPU) is a well-established parallelization domain to accelerate scientific and engineering computations in a number of fields. NVIDIA's Compute Unified Device Architecture (CUDA) is the most widely adopted programming model for GPGPU. In the research [34] [35], the hardware features of a GPU, and the process of developing optimized CUDA based code are discussed.

The pseudocode of the PRA implementation for GPGPU is shown in **Figure 4**. The first *for loop* implements the coarse propagator to compute the initial conditions in a sequential manner since it is an inexpensive computational task. These initial conditions computed by the coarse propagator are used in computing the fine solutions by the fine propagators executing in parallel. The first of the two inner *for loops* represents the fine propagators. The second of the two inner *for loops* represents the predictor-corrector to correct the coarse solution in a sequential manner.

For the GPGPU implementation, the sequential steps of the PRA are executed on the host (CPU) and the parallel step of the PRA executed on the device or accelerator (GPU). First, the coarse solutions computed on the host are copied from the host-to-device for use by the fine propagators. After the fine solutions are computed on the device, the fine solutions are copied back from device-to-host

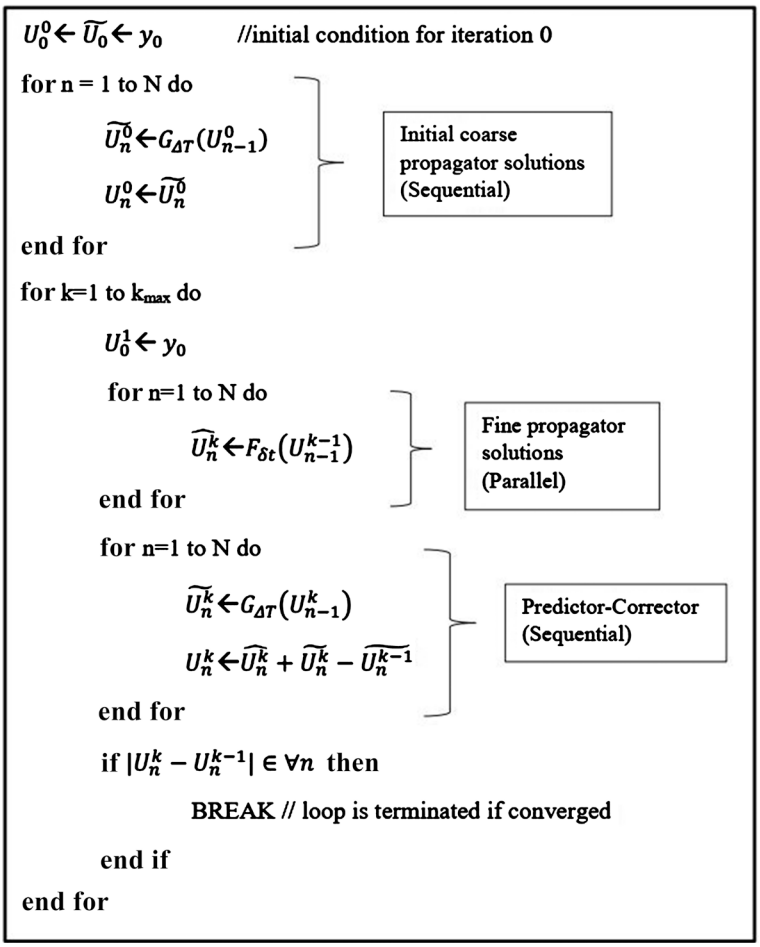


Figure 4. Pseudo code of PRA.

for the predictor-corrector step. The corrected coarse values on the host are again copied to the device for the next iteration of the fine propagators. Therefore, the memory transfers back and forth between host and device in each iteration contributes to an increase of the computation time. The focus of this work at this stage is on determining the suitability of PRA for solving ODEs on GPUs using CUDA, optimization techniques to reduce latency due to memory transfers and use of low latency shared and constant memories on the GPU are not addressed.

4.3. Test System

The performance of the PRA is demonstrated by studying the dynamics of a single machine infinite bus system (SMIB) shown in Figure 5. Power system studies are performed by considering the generator of interest connected to an infinite bus representing rest of the power grid. The dynamics of the generator in consideration due to a disturbance is studied using its mathematical model while the rest of the power grid is considered to be time-invariant to disturbances and thus modeled as an infinite bus.

The generator in Figure 5 is assumed to be delivering a constant power to the infinite bus, and a 3 phase to ground fault occurs in the middle of one of the

transmission lines connecting the generator bus-1 to the infinite bus-2. On fault occurrence, the rotor angle and the angular frequency of the generator will start changing with time and diverge from the infinite bus phase angle leading to instability of the system. In order to maintain stability, the fault should be cleared by isolating the faulted line from rest of the system within a short duration of time. This short duration of time is known as the critical clearing time. If the fault clearing time is less than the critical clearing time, the system will traverse to a new stable state otherwise the system becomes unstable. In our study, for both stable and unstable cases the generator ODEs solutions are computed using PRA and then compared with modified Euler's method to evaluate the performance of PRA.

The coefficients of the equations 12 through 17 of classical and detailed mathematical models of a generator are computed using the generator model parameters [33] [36] in Table 1 and Table 2, respectively.

In Figure 5, both of the transmission lines have a reactance of 0.3 pu while the transformer reactance is 0.2 pu.

5. Results and Performance Analysis

The PRA is implemented on a server having a Intel Xeon CPU E5-2670 @2.30 GHz, interfaced through the PCIe bus to with NVIDIA Quadro RTX 6000 GPU hosting 4608 computing cores with 24 GB GPU memory [37]. The C programming language version of CUDA is used in implementing the PRA for execution on GPUs.

TSA simulations using both classical and detailed generator models were performed using both the sequential algorithm and PRA. First, the variation of rotor angle of the generator with respect time due to a disturbance computed with

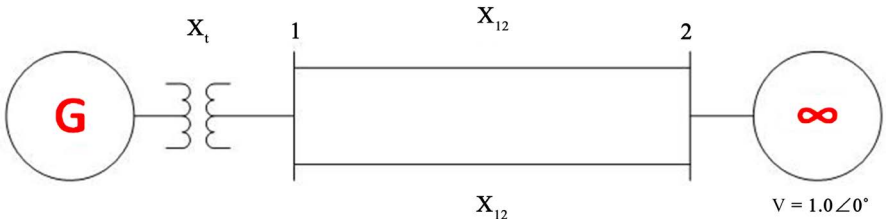


Figure 5. SMIB model.

Table 1. Generator parameters for classical model.

Parameter	H	X'_d
Numerical Values	5 MJ/MVA	0.3 pu

Table 2. Generator parameters for detailed model.

Parameter	H	X'_d	X'_q	X_d	X_q	T'_{do}	T'_{qo}
Numerical Values	3.74 MJ/MVA	0.23 pu	0.5 pu	1.93 pu	1.77 pu	5.2 s	0.81 s

traditional sequential and Parareal algorithms are compared to analyze the accuracy of PRA while satisfying a convergence tolerance of 0.01 radians or 0.57˚. Next, the impact of the number of coarse propagators and fine propagators on speedup is analyzed.

5.1. Simulations Using the Classical Generator Model

Classical generator model has only two state variables rotor angle δ and rotor speed ω that result in two ODEs that need to be solved at every time step. A 3φ to ground fault was simulated on one of the transmission lines of SMIB at time 0.5 secs and the fault is cleared at time 0.8 secs by isolating the faulted line from the rest of the SMIBs system. Since the ODEs are derived from the classical generator model, the TSA is performed to determine the first swing stability of the SMIB after experiencing a disturbance or fault. In **Figure 6(a)**, the variation of the rotor angle with time under pre-fault, during-fault and post-fault conditions are simulated using sequential and PRA are shown. The sequential simulation was implemented with a time step of 0.1 msec while time steps of 10 msecs and 0.1 msecs were used with the PRA coarse and fine propagators respectively. The maximum time step that could be used without the solution diverging for the sequential simulation was found to be 98 msecs and therefore the maximum coarse propagator stable time step is also 98 msecs. A three-phase to ground fault occurs at 0.5 secs and the fault is cleared within 0.3 secs. The rotor angle variation in **Figure 6(a)** approaches 90 degrees and then swings back indicating the SMIB is first swing stable for this particular fault with a fault clearing time of 0.3 secs. In **Figure 6(a)**, the rotor angles from PRA are the coarse propagator computed values which closely follows the rotor angle computed using the sequential algorithm. In **Figure 6(b)**, the absolute error between the rotor angles computed from the sequential and the PRA are shown. It can be seen that the maximum absolute rotor angle error is only 0.2˚ and the error has the same contour of the rotor angle variation. The small maximum error and the same contour demonstrate the accuracy of the PRA.

The second set of simulations was performed with a fault clearing time of 1.0 secs which is larger than the critical clearing time of 0.42 secs. Critical clearing time is the time before which a fault has to be cleared for the power system to transit to a stable steady state. Also, the critical clearing time is fault and system steady state dependent. In **Figure 7(a)**, the time domain solution of the rotor angle with sequential and PRA are shown. As expected, the rotor angle keeps increasing due to the system being unstable. The rotor angle computed using the PRA follows the angle from the sequential algorithm demonstrating the suitability of PRA even during the unstable system state. In **Figure 7(b)**, the absolute error between the sequential and PRA solution is presented. In **Figure 7(b)**, the magnitude of the error between the sequential and PRA is increasing while following the contour of the rotor angle variation. This is due to the numerical solutions computed by the coarse propagator are numerically instable (increasing

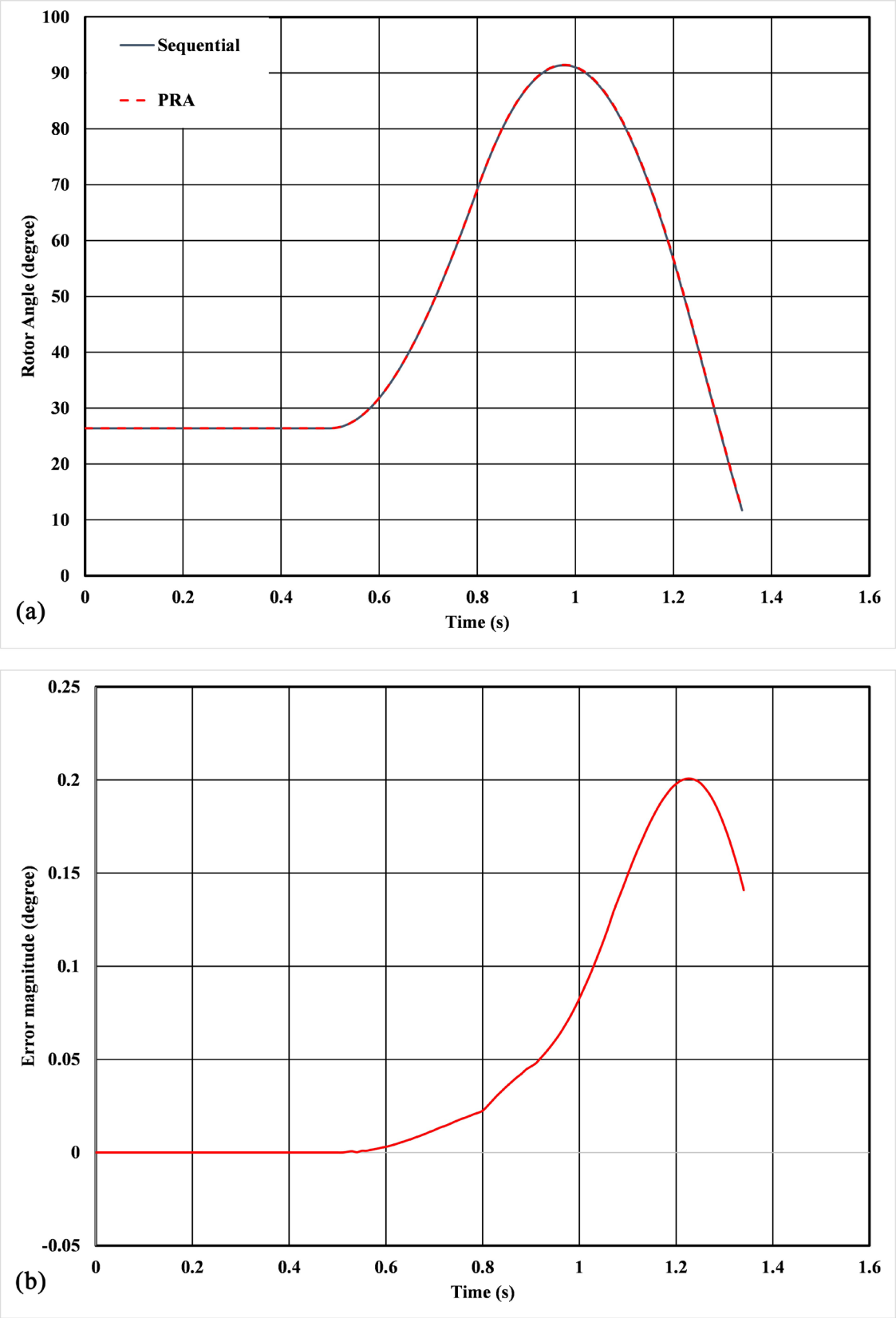


Figure 6. Rotor Angle Variation using Sequential and PRA of Classical Generator Model. (a) Rotor Angle variation with Sequential and PRA Simulations; (b) Rotor Angle Variation Error with PRA Simulations.

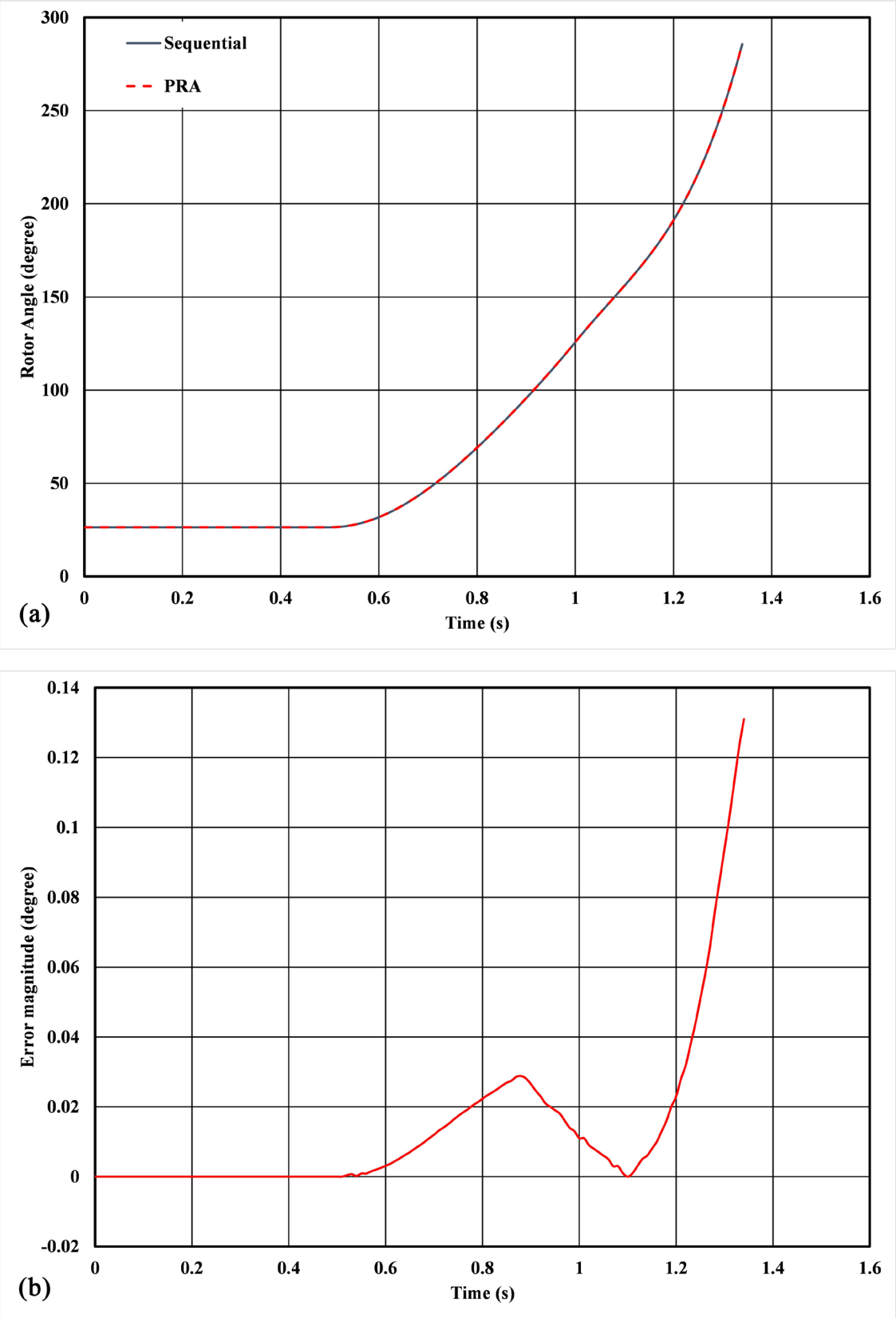


Figure 7. Time-domain simulation comparison of Rotor angle for unstable system. (a) Time-domain solution: Sequential v/s PRA for unstable system; (b) Absolute error between sequential and PRA for unstable system.

numerically). Furthermore, these numerically instable values are used as the initial conditions for fine propagators resulting in an amplification of the numerical instability. The numerical instability affects negatively the performance of the predictor-corrector stage of the PRA resulting in an increasing error. This behavior is expected as the system is unstable.

5.2. Simulations Using the Detailed Generator Model

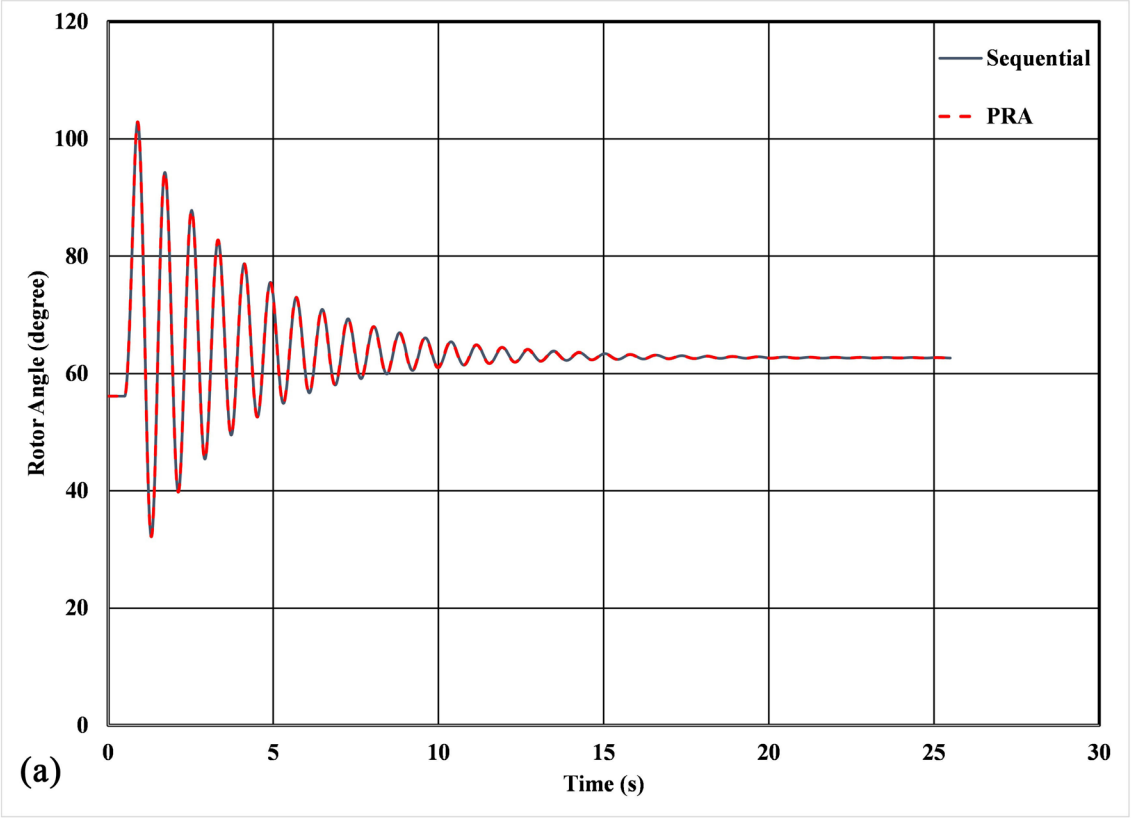
The ODEs of the detailed model of a generator incorporating the saliency and transient reactances are typically stiff compared to the ODEs of the classical model. Due to the stiffness, the maximum time step that could be used without the solution diverging for both the sequential simulation and the coarse-propagator was found to be 70 msecs compared to 98 msecs for the classical model. Therefore, the simulation parameters *i.e.*, the coarse and fine propagators time steps, the fault location and type, and the fault duration are identical to the simulations using the classical model. The simulations with the detailed model are carried out for a long period of time to study the effect of saliency and the damping.

In Figure 8(a), the rotor angle variations with sequential and PRA simulations are presented. The rotor angle solutions computed using the PRA is similar to the traditional sequential method. Also, it can be noticed that the rotor angle swing is damped and settles to a new system steady-state. In Figure 8(b), the rotor angle absolute error between the sequential and the PRA simulations is presented. The rotor angle computation with the detailed model involves the sequential solution of four ODEs at each time step and application of the predictor-corrector on all four ODEs at the end of each fine propagator iteration. The numerical values of the solutions of the four ODEs during the initial phase of fault are large. These numerical large values cascade through the four ODEs within a fine propagator iteration and due to the cascading effect, the numerical error is large initially as shown in Figure 8(b). After the clearing of the fault, and due to damping the rotor angle swing reduces and correspondingly the numerical values resulting in smaller numerical error between the sequential and PRA simulations.

In Figure 9(a), the rotor angle variation with time when the fault clearing time is 1.3 secs is shown. The 1.3 secs are larger than the 0.77 secs critical clearing time and therefore the system is unstable. The rotor angle variations computed using the PRA follows that computed using the sequential algorithm. In Figure 9(b), the absolute error between PRA solution and sequential solution is shown. The absolute error has larger value due to the cascading of the error through the four ODEs.

5.3. Performance Analysis

The performance of PRA is analyzed using the execution time speedup achieved with respect to the traditional sequential algorithm. The speedup is given by Equation (21).



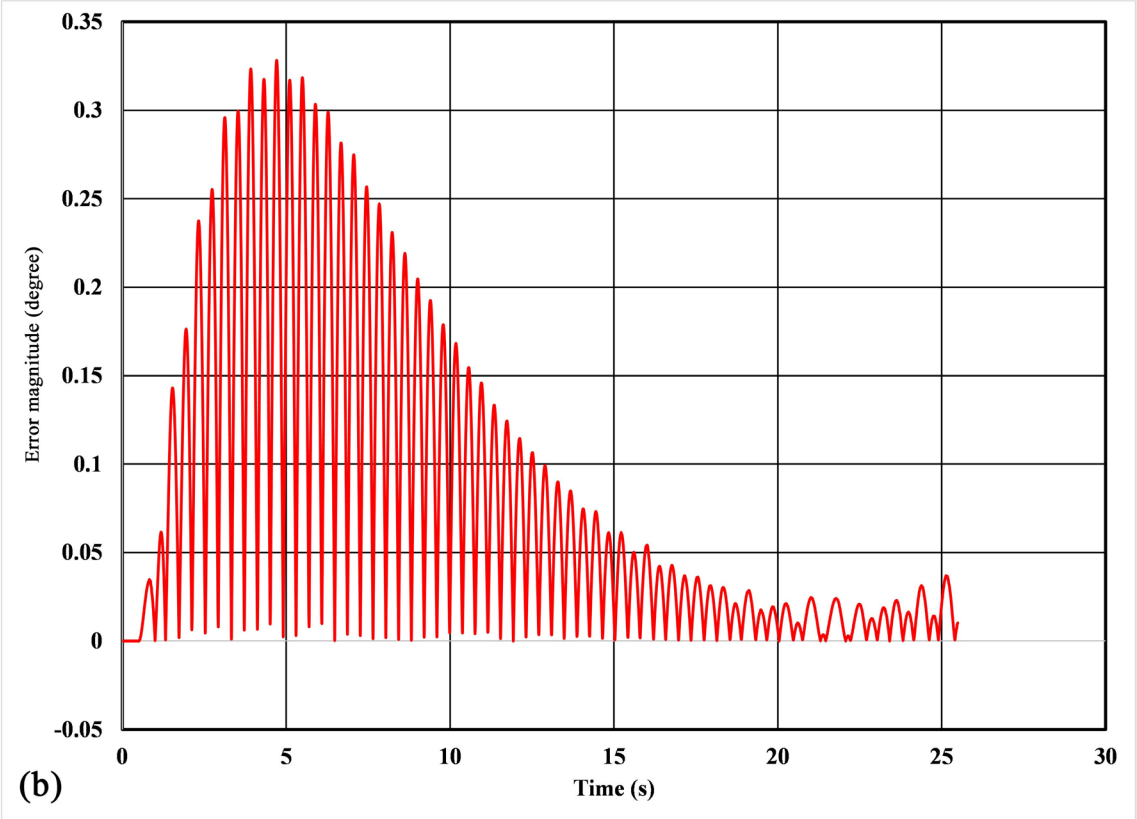


Figure 8. Time-domain simulation comparison of rotor angle for detailed model. (a) Time-domain Solution: Sequential v/s PRA; (b) Absolute error between sequential and PRA.

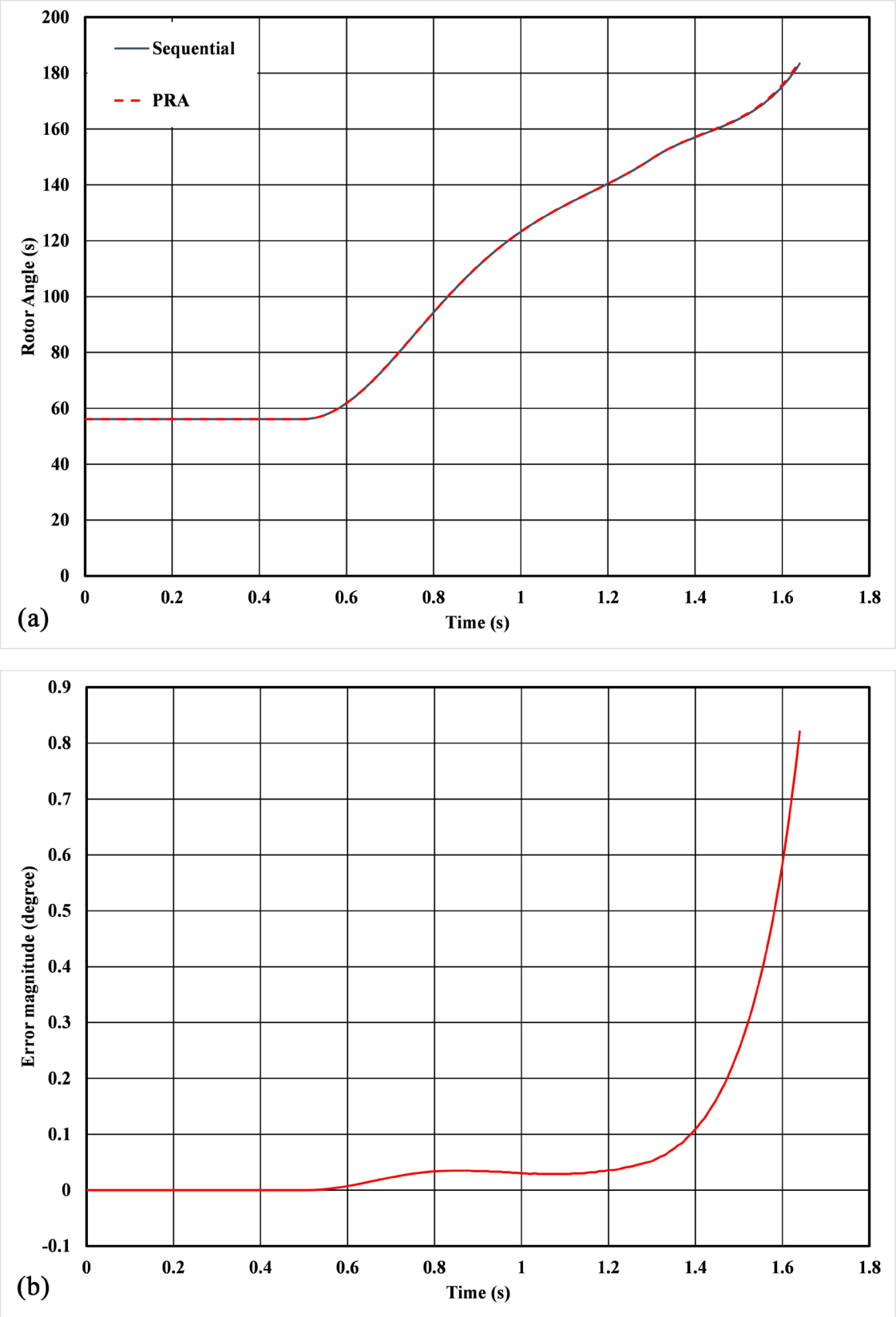


Figure 9. Time-domain simulation comparison of Rotor angle for unstable system. (a) Time-domain solution: Sequential v/s PRA for unstable system; (b) Absolute error between sequential and PRA for unstable system.

$$\text{speedup} = \frac{T_{seq}}{T_{PRA}} \tag{21}$$

where,

T_{seq} is the computation time of the sequential algorithm.

T_{PRA} is the execution time of the PRA.

The T_{PRA} is defined as the execution time since it is the sum of four-time components as shown in Equation (22)

$$T_{PRA} = t_H^c + \sum_{i=1}^{N}\left(t_H^G + t_G^f + t_G^H + t_H^{pc}\right) \tag{22}$$

where,

t_H^c is the computation time of the coarse propagator on the host.

t_H^G is the memory transfer latency between the host and the GPU.

t_G^f is the computation time of the fine propagators on the GPU.

t_G^H is the memory transfer latency between the GPU and the host.

t_H^{pc} is the computation time of the predictor-corrector on the host.

N is the number of iterations.

The coarse propagator computation time is dependent on the coarse propagator time step t_{step}^c and the fixed interval of time T for which the ODEs are solved. For a fixed T, the coarse propagator computational time will increase with smaller t_{step}^c. The memory transfer latencies t_H^G and t_G^H both are also dependent on the coarse propagator time step t_{step}^c. The number of fine propagators N^f corresponding to a coarse propagator time step t_{step}^c and for a given T is

$$N^f = \frac{T}{t_{step}^c} \tag{23}$$

By varying N^f, the number of threads executing in parallel on the GPU cores is varied and varying the fine propagator time step t_{step}^f the computation load of each thread is varied.

The speedup achieved using the PRA is demonstrated through a number of simulations with varying t_{step}^c or N^f, and t_{step}^f. In Table 3, the execution times of both sequential algorithm and PRA computing the solutions of the ODEs of the classical model along with the speedups are presented. The simulation time T was set to 3.06 secs to account for the first swing stability with classical model of the generator. The execution time of the PRA executing on the GPU is significantly less compared to sequential algorithm computation time

Table 3. PRA execution time and speedup with classical model.

t_{step}^c (msecs)	t_{step}^f (usecs)	N^f	Execution time (msecs)		Speedup
			Sequential	CUDA	
20.0	200.0	128	1.778	0.183	9.7
10.0	100.0	256	3.594	0.242	15
5.0	50.0	512	7.105	0.283	25

resulting in a speedup of 25×. The PRA on GPU provides better performance when the fine propagator computation load is large, *i.e.* smaller t_{step}^{f}.

In Figure 10, the variation of speedup with N^{f} is shown. Since the speedup is increasing linearly, the parallel scalability of the PRA has strong scaling efficiency. The strong scaling efficiency is due to the t_{G}^{f} being significantly large compared to the sum of remaining four time components in Equation (22).

In Table 4, the execution times of both sequential algorithm and PRA computing the solutions of the ODEs of the detailed model along with the speedups are presented. The simulation time T was set to 26.1 secs to account for the long term dynamic stability with the detailed model of a generator. In Table 4, it can be seen that the PRA execution time is significantly small compared to the sequential computational time resulting in a speedup of 31×. It is important to emphasis that with the detailed model, the number of ODEs solved sequentially

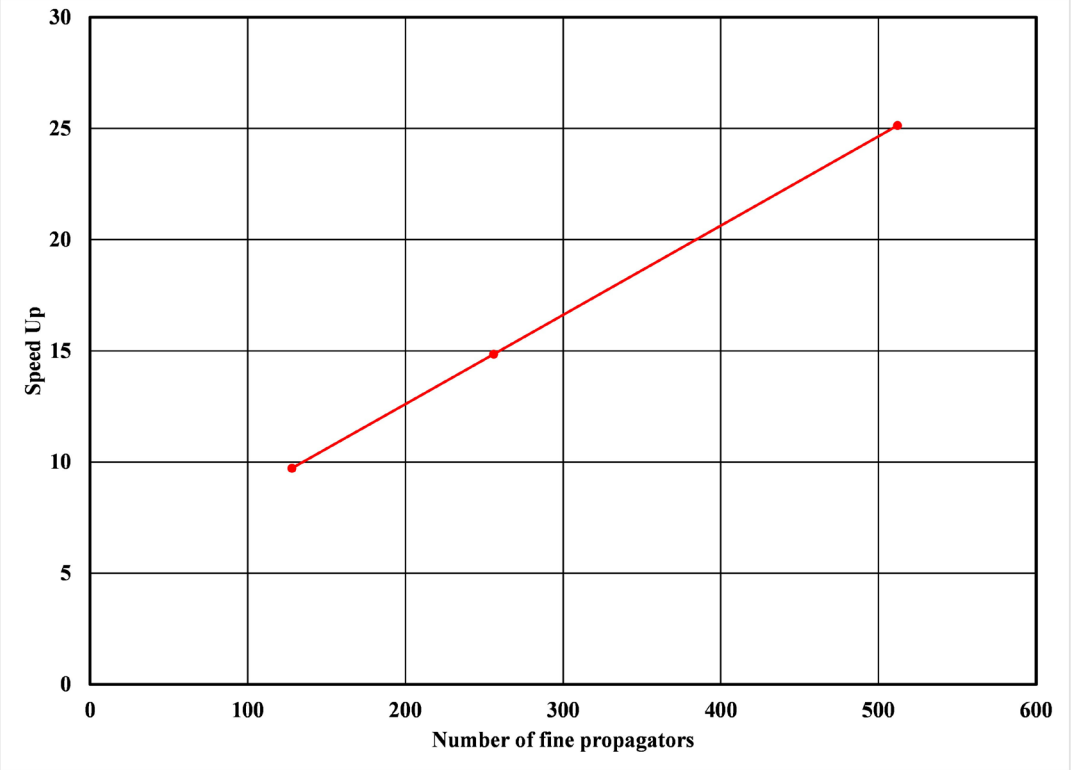


Figure 10. Variation of Speedup with N^{f} for Classical model.

Table 4. Execution time and speedup for detailed model.

t_{step}^{c} (msecs)	t_{step}^{f} (usecs)	N^{f}	Execution time (msecs)		Speedup
			Sequential	CUDA	
10.0	100.0	2560	40.786	1.875	21.7
5.0	50.0	5120	72.57	2.728	26.6
2.0	20.0	12800	159.665	5.043	31.7
1.0	10.0	25600	289.96	9.714	30.0

at each time step is twice that with the classical model.

In Figure 11, the variation of speedup with N^f is shown. In Figure 11, it can be seen that the speedup does not increase linearly and flattens with increasing N^f indicating the parallel scalability has a weak scaling efficiency. The weak scaling is due to the sum of the coarse propagator computation time t_H^c and the memory transfer latencies (t_H^G, t_G^H) being larger compared to the fine propagators computation time t_G^f. The t_H^c is large due to four ODEs solved at each time step and larger memory transfer latencies to transfer the larger coarse propagator solutions from host to GPU and vice versa. The performance of the PRA with the detailed model is memory bound.

Therefore, from Figure 11, it is evident that the performance of the PRA algorithm decreases as higher level models with larger number of differential equations are implemented to study the dynamic stability. However, the execution time of the PRA is still significantly small compared to the computational time of the sequential algorithm demonstrating the suitability of PRA for near real-time transient stability analysis.

6. Conclusion

TSA performed using the time-domain solution approach is a compute-intensive problem and is typically conducted offline by the utilities. In this paper, the use of PRA to solve the ODEs for two synchronous generators models of a SMIB test system to perform TSA using GPUs has been demonstrated successfully. The

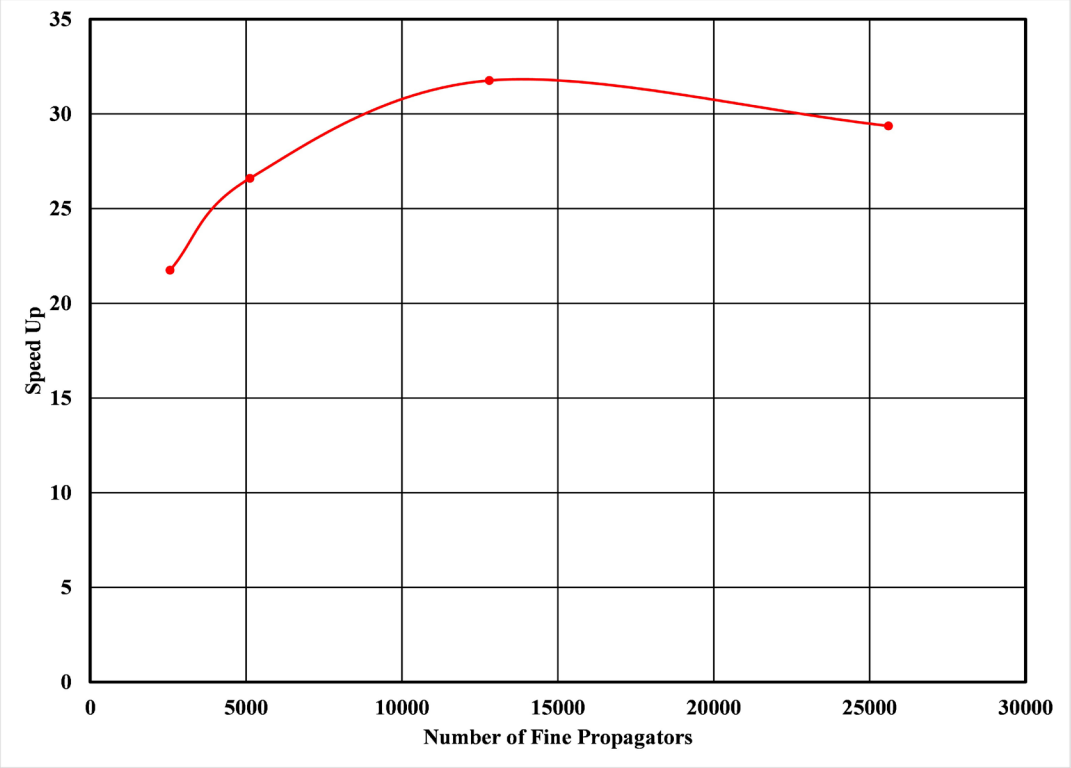


Figure 11. Variation of Speedup with N^f for Detailed model.

PRA was evaluated for accuracy with both stable and unstable cases of the test system. The absolute error between the ODE solutions by PRA and the sequential algorithm is very small demonstrating the accuracy of the PRA. The PRA speedup achieved using GPUs demonstrated that the numerical integration computational time can be significantly reduced in comparison to traditional sequential numerical integration. However, PRA is an iterative algorithm that can impact the performance due to significant amount of memory transfers between the host and device for systems with higher-order generator models. In future work, various methods will be explored to mitigate the memory transfers between the host and device, and the PRA algorithm will be tested for higher-order generator models for large power systems.

Acknowledgements

This work was supported in part by the Department of Energy under grant DE-SC0012671.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Tylavsky, D., Bose, A., Alvarado, F., Betancourt, R., Clements, K., Heydt, G.T., Huang, G., Ilic, C., La Scala, M., Pai, M.A. and Pottle, C. (1992) Parallel Processing in Power Systems Computation. *IEEE Transactions on Power Systems*, **7**, 629-638. https://doi.org/10.1109/59.141768

[2] La Scala, M., Bose, A., Tylavsky, D.J. and Chai, J.S. (1990) A Highly Parallel Method for Transient Stability Analysis. *IEEE Transactions on Power Systems*, **5**, 1439-1446. https://doi.org/10.1109/59.99398

[3] La Scala, M., Sblendorio, G., Bose, A. and Wu, J.Q. (1996) Comparison of Algorithms for Transient Stability Simulations on Shared and Distributed Memory Multiprocessors. *IEEE Transactions on Power Systems*, **11**, 2045-2050. https://doi.org/10.1109/59.544683

[4] Granelli, G.P., Montagna, M., La Scala, M. and Torelli, F. (1993) Relaxation-Newton methods for Transient Stability Analysis on a Vector/Parallel Computer. *Conference Proceedings Power Industry Computer Application Conference*, Scottsdale, AZ, 4-7 May 1993, 387-393. https://doi.org/10.1109/PICA.1993.290991

[5] Shu, J., Xue, W. and Zheng, W. (2005) A Parallel Transient Stability Simulation for Power Systems. *IEEE Transactions on Power Systems*, **20**, 1709-1717. https://doi.org/10.1109/TPWRS.2005.857266

[6] Esmaeili, S. and Kouhsari, S.M. (2007) A Distributed Simulation Based Approach for Detailed and Decentralized Power System Transient Stability Analysis. *Electric Power Systems Research*, **77**, 673-684. https://doi.org/10.1016/j.epsr.2006.06.008

[7] Crow, M.L. and Ilic, M. (1990) The Parallel Implementation of the Waveform Relaxation Method for Transient Stability Simulations. *IEEE Transactions on Power Systems*, **5**, 922-932. https://doi.org/10.1109/59.65922

[8] Morales, F., Rudnick, H. and Cipriano, A. (2001) Electromechanical Transients Simulation on a Multicomputer via the VDHN-Maclaurin Method. *IEEE Transactions on Power Systems*, **16**, 418-426. https://doi.org/10.1109/59.932277

[9] Dufour, C., Jalili-Marandi, V., Bélanger, J. and Snider, L. (2012) Power System Simulation Algorithms for Parallel Computer Architectures. 2012 *IEEE Power and Energy Society General Meeting*, San Diego, CA, 22-26 July 2012, 1-6. https://doi.org/10.1109/PESGM.2012.6344986

[10] Nievergelt, J. (1964) Parallel Methods for Integrating Ordinary Differential Equations. *Communications of the ACM*, **7**, 731-733. https://doi.org/10.1145/355588.365137

[11] Alvarado, F.L. (1979) Parallel Solution of Transient Problems by Trapezoidal Integration. *IEEE Transactions on Power Apparatus and Systems*, **PAS-98**, 1080-1090. https://doi.org/10.1109/TPAS.1979.319271

[12] Chai, J.S. and Bose, A. (1993) Bottlenecks in Parallel Algorithms for Power System Stability Analysis. *IEEE Transactions on Power Systems*, **8**, 9-15. https://doi.org/10.1109/59.221242

[13] Wang, F.Z. (1998) Parallel-in-Time Relaxed Newton Method for Transient Stability Analysis. *IEE Proceedings-Generation, Transmission and Distribution*, **145**, 155-159. https://doi.org/10.1049/ip-gtd:19981836

[14] Nielsen, A.S. (2012) Feasibility Study of the Parareal Algorithm. Doctoral Dissertation, Technical University of Denmark, Denmark.

[15] Maday, Y. (2008) The Parareal in Time Algorithm. https://doi.org/10.1063/1.3241386

[16] Baffico, L., Bernard, S., Maday, Y., Turinici, G. and Zérah, G. (2002) Parallel-in-Time Molecular-Dynamics Simulations. *Physical Review E*, **66**, Article ID: 057701. https://doi.org/10.1103/PhysRevE.66.057701

[17] Staff, G.A. and Rønquist, E.M. (2005) Stability of the Parareal Algorithm. In: *Domain Decomposition Methods in Science and Engineering*, Springer, Berlin, Heidelberg, 449-456. https://doi.org/10.1007/3-540-26825-1_46

[18] Gander, M.J. and Hairer, E. (2008) Nonlinear Convergence Analysis for the Parareal Algorithm. In: *Domain Decomposition Methods in Science and Engineering XVII*, Springer, Berlin, Heidelberg, 45-56. https://doi.org/10.1007/978-3-540-75199-1_4

[19] Harden, C.R. (2008) Real Time Computing with the Parareal Algorithm. Doctoral Dissertation, Florida State University, Tallahassee, FL.

[20] Ruprecht, D. and Krause, R. (2012) Explicit Parallel-in-Time Integration of a Linear Acoustic-Advection System. *Computers & Fluids*, **59**, 72-83. https://doi.org/10.1016/j.compfluid.2012.02.015

[21] Minion, M. (2011) A Hybrid Parareal Spectral Deferred Corrections Method. *Communications in Applied Mathematics and Computational Science*, **5**, 265-301. https://doi.org/10.2140/camcos.2010.5.265

[22] Berry, L.A., Elwasif, W., Reynolds-Barredo, J.M., Samaddar, D., Sanchez, R. and Newman, D.E. (2012) Event-Based Parareal: A Data-Flow Based Implementation of Parareal. *Journal of Computational Physics*, **231**, 5945-5954. https://doi.org/10.1016/j.jcp.2012.05.016

[23] Staff, G. (2003) Convergence and Stability of the Parareal Algorithm: A Numerical and Theoretical Investigation.

[24] Bal, G. and Maday, Y. (2002) A "Parareal" Time Discretization for Non-Linear

PDE's with Application to the Pricing of an American Put. In: *Recent Developments in Domain Decomposition Methods*, Springer, Berlin, Heidelberg, 189-202. https://doi.org/10.1007/978-3-642-56118-4_12

[25] Maday, Y. and Turinici, G. (2003) Parallel in Time Algorithms for Quantum Control: Parareal Time Discretization Scheme. *International Journal of Quantum Chemistry*, **93**, 223-228. https://doi.org/10.1002/qua.10554

[26] Gurrala, G., Dimitrovski, A., Pannala, S., Simunovic, S. and Starke, M. (2015) Parareal in Time for Fast Power System Dynamic Simulations. *IEEE Transactions on Power Systems*, **31**, 1820-1830. https://doi.org/10.1109/TPWRS.2015.2434833

[27] Duan, N., Dimitrovski, A., Simunovic, S. and Sun, K. (2016) Applying Reduced Generator Models in the Coarse Solver of Parareal in Time Parallel Power System Simulation. 2016 *IEEE PES Innovative Smart Grid Technologies Conference Europe*, Ljubljana, Slovenia, 9-12 October 2016, 1-5. https://doi.org/10.1109/ISGTEurope.2016.7856184

[28] Duan, N., Dimitrovski, A., Simunovic, S., Sun, K., Qi, J. and Wang, J. (2018) February. Embedding Spatial Decomposition in Parareal in Time Power System Simulation. 2018 *IEEE Power & Energy Society Innovative Smart Grid Technologies Conference*, Washington DC, 19-22 February 2018, 1-6. https://doi.org/10.1109/ISGT.2018.8403389

[29] Xia, S., Bu, S., Hu, J., Hong, B., Guo, Z. and Zhang, D. (2018) Efficient Transient Stability Analysis of Electrical Power System Based on a Spatially Paralleled Hybrid Approach. *IEEE Transactions on Industrial Informatics*, **15**, 1460-1473. https://doi.org/10.1109/TII.2018.2844298

[30] Cheng, J., Grossman, M. and McKercher, T. (2014) Professional Cuda C Programming. John Wiley & Sons, New York.

[31] Wang, B. and Sun, K. (2015) Power System Differential-Algebraic Equations. arXiv Preprint arXiv:1512.05185.

[32] Kundur, P., Balu, N.J. and Lauby, M.G. (1994) Power System Stability and Control. Volume 7, McGraw-Hill, New York.

[33] Padiyar, K.R. (1996) Power System Dynamics: Stability and Control. John Wiley, New York.

[34] Kumar, R. Muknahallipatna, S. and McInroy, J. (2016) An Approach to Parallelization of SIFT Algorithm on GPUs for Real-Time Applications. *Journal of Computer and Communications*, **4**, 18-50. https://doi.org/10.4236/jcc.2016.417002

[35] Ramakrishnaiah, V.B., Muknahallipatna, S. and Kubichek, R.F. (2017) Adaptive Region Construction for Efficient Use of Radio Propagation Maps. *Journal of Computer and Communications*, **5**, 21-51. https://doi.org/10.4236/jcc.2017.58003

[36] Tanwani, N.K., Memon, A.P., Adil, W.A. and Ansari, J.A. (2014) Simulation Techniques of Electrical Power System Stability Studies Utilizing Matlab/Simulink. *Engineer*, **9**, 18.

[37] Quadro RTX 6000 GPU. https://www.nvidia.com/en-us/design-visualization/quadro/rtx-6000/