This paper describes an open source power system simulation tool: Resilient Adaptive Parallel sImulator for griD (RAPID), a package of Python codes that implements an advanced power system dynamic simulation framework. The main function of RAPID is time-domain simulation, and it contains details of the solution process, including the quasi-steady-state power flow analysis, to solve nonlinear differential algebraic equations describing the detailed representation of network and transient stability models of dynamic devices in a computationally efficient manner. In particular, RAPID utilizes and incorporates emerging solution techniques; a novel “parallel-in-time” (Parareal) algorithm, adaptive model reduction, and semi-analytical solution methods as well as the standard numerical solution methods. Moreover, the whole simulation process for the transmission network has been coupled with OpenDSS, a widely used open-source distribution system simulator, to enable the co-simulation of integrated transmission and distribution systems. Basic structure and features of RAPID are presented to provide an easy-to-understand guideline for the usage of the tool and illustrate its unique capabilities. This paper also presents simulation results for a number of test scenarios and demonstrates RAPID’s values for advancing power systems research in dynamic modeling and simulation techniques toward researchers as well as educators and students.

INDEX TERMS High-performance computing, open-source tools, parallel algorithm, power engineering research, power systems, power system co-simulation, power system dynamic, semi-analytical solution methods, time-domain simulation.

I. INTRODUCTION

With increasing grid complexity resulting from the ongoing grid modernization efforts, studies of dynamic behavior are becoming more important, which also renders research on improved simulation of dynamic behavior very important [1]. One of key concerns is to improve the computational performance of time-domain simulation, solving a large number of nonlinear differential algebraic equations (DAEs), for future electric grids whose components are governed by faster and more complex dynamics (e.g., [2]). An added concern is the increasing interaction between transmission and distribution (T&D) systems with more interconnections and integration of distributed energy resources (DERs) within the transmission grid, distribution networks, and behind-the-meter (BTM) [3], [4]. A key enabler for promoting and advancing research in this research area is to establish open source software tools developed by a popular high-level scientific computing language.

To serve the aforementioned objective, this paper describes RAPID (Resilient Adaptive Parallel sImulator for griD), a new open source Python-based power system dynamic simulation tool, which is freely available online [5]. RAPID is a python package designed to solve DAEs in a systematic way and keep the codes simple for easy understanding and customizing from end users’ perspective while giving the best possible performance. In the last decade, Python has...
become a popular scientific computing tool for both research and educational purposes as well as industries. Key benefits facilitated by using Python are simple programming syntax (i.e., combining a high-level language), code readability, and complimentary for use and distribution along with a rich set of libraries and frameworks, mature and large community, and versatility allowing the integration with other languages and tools. It is most suitable for developing an open source software package for power systems research.

In the context of software packages for power system dynamic simulations, commercial software simulators (e.g., PSS/E and PowerWorld) provide a well-tested solution and are typically designed to offer many functions with an all-in-one and easy-to-use philosophy for end users. However, this completeness may not be the most crucial aspect and result additional challenges for research and educational purposes; the source codes are often closed, which disallows users from changing the codes or adding new algorithms. To this end, several open source research tools in Matlab environment are developed (e.g., PSAT [6], PST [7], and MATPOWER [8]), which keeps balance between the performance and the flexibility/ability to customize the codes for one’s own research. MATPOWER is one successful example widely used particularly for the steady-state analysis without the time-domain simulation. Aforementioned two others provide the time-domain simulation function with very classic yet standard techniques for the solution process of nonlinear DAEs.

In Python environment, initial and recent efforts for developing open source research tools include PyPower (no longer actively maintained) and Pandapower [9] whose features include power system modeling, analysis, and optimization for the steady-state analysis without the time-domain simulation. Another recent Python-based open source research tool includes the work [10] that aims at improving the modeling procedure of DAEs in power systems by utilizing the help of a symbolic toolbox. These efforts show a clear potential of Python-based open source tools and also suggest value in the continuous development beyond the steady-state analysis to be useful for researchers and educators.

Motivated by these, the Python package of RAPID focuses on the time-domain simulation and provides a library of functions, which details the solution process of nonlinear DAEs from modeling of the network (e.g., \( Y_{bus} \) matrix) and fault scenarios, calculating initial conditions with the power flow analysis, and solving nonlinear DAEs. Note that RAPID leverages MATPOWER and Pandapower for the power flow analysis. Therefore, the library at this lower level offers great versatility to modify and customize the code. At a higher level, the structure of the code is designed for an easy understanding and extension to allow the addition of user-defined functions. Moreover, other unique capabilities of RAPID include (1) the incorporation of emerging solution techniques to improve its computational performance and (2) an innovative co-simulation framework, integrated with the well-established open source distribution system simulator (i.e., OpenDSS). In particular, it implements a novel “parallel-in-time” (Parareal) algorithm [11], [12], [13], [14], semi-analytical solution (SAS) methods [15], [16], [17], [18], [19], and adaptive model reduction techniques [20], [21], [22] as well as transmission and distribution (T&D) co-simulation [23] in the context of emerging solution techniques. While these advanced techniques have been widely applied in other applied sciences and engineering (e.g., [24] for the SAS methods and [25], [26] for the Parareal algorithm), these have been very lightly explored in the power engineering community. One primary reason is the requirement of additional expertise and steep learning curve on applied mathematics and parallel programming techniques that are outside of power engineering expertise; each of these requires significant efforts and complex programming skills to understand techniques and facilitate tailored power system applications, respectively. These challenges also apply to the design of large scale co-simulation frameworks for the integrated T&D networks.

Therefore, RAPID can reduce this barrier significantly and be the first milestone for other interested researchers to explore such emerging solution techniques in predicting the dynamic behavior of future power system networks. As a result, this open source toolbox has a great potential to promote and accelerate R&D in dynamic modeling and simulation techniques as well as testing and developing a wide range of innovative control and protection algorithms that would coordinate large amounts of local and wide-area measurements for improving reliability, resiliency, and security. In addition to research value, a wide readership in the power industry would benefit from this advanced open source grid simulator.

The rest of the paper is organized as follows. Section II describes the main features and structure of RAPID. Section III reviews power system modeling for the time-domain simulation as well as the T&D co-simulation framework. Section IV details the solution approach and algorithms. Simulations and numerical case studies are summarized in Section V. Conclusions are presented in Section VI.

II. OVERVIEW OF RAPID
A. HIGH PERFORMANCE COMPUTING
1) APPLICATIONS IN POWER SYSTEM NETWORKS
Current power systems have mainly relied on the sequential computation framework on single-processor workstations to simulate its dynamic behaviors. Given the growing complexity of the modern power grids, the sequential computation frameworks may not be capable of tackling larger and ever-growing problems that future power systems face. In this aspect, high performance computing (HPC) has a great potential to address such computational challenges by improving the computational performance of the time-domain simulation and thus enable predicting power system dynamic behaviors in real time. Indeed, the need for HPC architectures in power system applications has long been recognized [27];
several research efforts have discussed parallel computations for such power system problems as state estimation, massive contingency analysis, and multiple dynamic simulations.

Despite potential capabilities and research interests, the power system dynamic simulations have not widely utilized HPC in research and practice. One primary reason is related to the complexity and high cost of understanding, developing, and implementing parallel algorithms [28]. It requires strong parallel programming skills in addition to power engineering expertise, and a resulting outcome is typically a one-time implementation that is very complex to understand and only usable by a developer of that simulator. In addition, such research efforts are not easily and freely available to other researchers. The incorporation of other emerging techniques (e.g., SAS methods and co-simulation framework) also follow similar challenges. Consequently, all of these barriers hinder investigating and enjoying such emerging solution techniques to improve the computational performance of the time-domain simulation. Hence, the provision of highly flexible software packages as an open source tool on commonly used and popular scientific computing language is highly useful to lower the initial barrier to examine potentials of various emerging solution techniques for power system dynamic simulations as well as other applications (e.g., control and protection algorithms), which corresponds to the main goal of RAPID.

2) PARALLEL COMPUTING IMPLEMENTATION
To program parallel computing, RAPID utilizes the message passing interface (MPI) program. MPI is a communication protocol and portable message-passing standard designed to function on parallel computing architectures [29], defining how concurrent processes can communicate and thus cooperate to complete a given task in a parallel manner and shorter time. In particular, MPI provides a high performance, scalable, and portable way to express parallel algorithms. The main source code of RAPID illustrates the usage of MPI to implement the Parareal algorithm in Python environment.

B. OUTLINES
RAPID has been developed to be portable and open source using Python, which is available for running on the common operating systems, e.g., Windows, Linux, and Mac OS X. The overall scheme of RAPID is pictorially represented in Fig. 1. RAPID takes fault definition, data files (i.e., test systems), and Parareal setting as inputs to run the time-domain simulation. Initially, the power flow algorithm with the user defined dynamic model is solved for the initialization. Then, depending on the setting, RAPID solves DAEs using either the standard sequential approach or Parareal algorithm. As shown, various solution methods have been included in RAPID; the Runge-Kutta 4th order (RK4) method, midpoint-Trapezoidal predictor-corrector (Trap) method [12], Adomian decomposition method (ADM) [18], Homotopy Analysis method (HAM) [18], and adaptive model reduction (AMR) method [20]. In addition, for the ADM and HAM, RAPID employs multi-stage approach, which applies ADM and HAM over multiple intervals of time, to improve the convergence region of the ADM and HAM; this is referred to
as the multistage ADM (MADM) and the multi-stage HAM (MHAM). Furthermore, RAPID provides the interface to OpenDSS, which highly extends its capability to capture the dynamic interplay between the transmission and distribution network.

C. COMPARISON OF RAPID WITH OTHER RELEVANT WORKS

There have been efforts developing open-source parallel simulation tools for power systems utilizing capabilities of supercomputers with HPC clusters [30], [31], [32]. For example, a joint effort reported in [30] has used a supercomputer having 1,296 computer nodes (16 cores per node) to simulate 40,906 contingencies on an Eastern Interconnection power system model with more than 25,000 buses. The average time spent on simulating a single contingency is less than the simulation window so as to achieve “faster-than-real-time (FTRT) simulation” at the interconnection level [30], [31]. Another effort [32] aimed to speed up a single simulation run by HPCs on a Western Interconnection power system model. However, these works mainly focused on enhancement and parallelization of conventional power system simulation engines (e.g., it only employs the simplified generator model, and the parallelization is mainly at the contingency level, which is relatively straightforward to distribute the task). It is very difficult to achieve “faster-than-real-time simulation” on each contingency for the worst case with detailed generator models and control devices. To the best of our knowledge, this work is the first known parallel simulation approach that can achieve such an ambitious goal.

D. OPEN SOURCE TOOL OTHER THAN PYTHON IN POWER SYSTEMS

In recent year, there have been a few efforts to develop open-source simulation tools other than Python. These include Julia-based tools [33], [34] and C++-based tools [35], [36]. Python may share some similarity with Julia in terms of syntax, but has different features as compared to C++. Python has become popular in the scientific community due to its simple, concise, and easily-readable syntax, which allows for rapid code prototyping and cross-platform development. Because our focus is on efficient algorithms and not necessarily on efficient code, we opted for Python as more relevant for these purposes. On the other hand, C++ language being at a lower programming level may be more complicated to read with a lengthier code. We are aware that C++ has the capability for better performance and faster execution speed, but this also depends on the skills of the individual programmer. Therefore, we have left the C++ translation to professional programmers for embedded or enterprise applications. The reader may find relevant discussions in [37] for comparative analysis and [38], [39] for the differences between Python and C++ from a learning perspective.

E. RUNNING RAPID

To run RAPID, the user needs to define arguments: key arguments and optional arguments. Here, the key arguments are tstart and tend. For example, RAPID can be run from the command line with the following arguments:

```
mpiexec -n 20 python para_real.py 0 1  
-nCoarse 25 -nFine 250  
-tol 0.01 -tolcheck maxabs  
-debug 1 -fault fault.json  
-o result.csv
```

This runs RAPID for the 1s simulation period (i.e., tstart is 0 and tend is 1) on 20 processors, which corresponds to 20 subintervals, with defined time increments (i.e., 25 time increments for the coarse operator in each of 20 subintervals and 250 time increments for the fine operator in each of 20 subintervals), using tolerance of 0.01 for maximum absolute value of the difference between iterations, reads fault scenarios defined in the fault.json file, and then saves solutions into the result.csv file. The usage of the command mpiexec -n 20 is to run the MPI program with 20 processors. Further details can be found in the manual available online [5].

III. DESIGN CONSIDERATIONS IN RAPID

RAPID is designed to highly flexible such that is allows users to easily understand, modify, and add functions and/or data for one’s own research purpose.

A. DATA FORMAT

The data of test systems is required to run RAPID. This input data is defined in the MAT-file of MATLAB, and each data contains the dynamic data and network data.

1) DYNAMIC DATA

the dynamic data contains generator data (GenData), excitation data (ExcData), turbine governor data (TurbData), and generator saturation data (satData). More details regarding the dynamic data format can be found [5].

2) NETWORK DATA

the network data contains the network information (mpc), which follows the format of MATPOWER [8]. This data is used to construct the admittance matrix (Ybus) and solve the power flow problem. The system structure is specified by mpc.bus, mpc.branch, and mpc.gen.

With this data format, the user can consider a wide range of IEEE test networks provided from MATPOWER and easily modify the dynamic and network data. The next version will include the data conversion process such that the input data in any form (e.g., the .raw file widely used for PSS/E) can be converted to the MAT-file of MATLAB to be used in RAPID.
B. MODELING OF DAEs
In this paper, the models of synchronous generators, exciters, turbine governors, and saturation effects are considered in the RAPID framework. To include other devices such as nonlinear loads, renewable energy generators, and relevant control devices in RAPID, one can follow the similar approach used for the models of synchronous generators as an example. That is, these new devices are usually modeled as a set of ordinary differential equations (ODE) or DAE models. Therefore, they can be easily added to the major component of RAPID framework. In a simplified way, adding new device models to RAPID is straightforward because the associated ODE or DAE models with these new devices could be directly coded to extend or replace the original ODE or DAE functions (detailed in fn.py) in RAPID tool. Thus, one can add new DAEs and/or replace original DAEs with the python file fn.py. Note that adding new DAEs and/or replacing original DAEs implies changes in the initialization step which can be also easily modified in the python file initial.py. Other major components (e.g., adaptive model reduction, and SAS methods) demand additional efforts such as the linear approximation for DAEs and the derivation of SAS terms, but they can all be done by modifying functions in fn.py.

C. FAULT MODELING
To provide a higher flexibility for defining a wide range of faults, RAPID utilizes the json module that the user can define faults in detail with many different attributes (e.g., fault type, start time, and end time) and interfaces with Python. In the fault.json file, the user can define the start and end time for the fault, the fault type with three choices (bus, line, generator), the index (e.g., “index”: [1] means the 3-phase fault at bus 1) for that fault type, and the presence of line trip after that fault (e.g., “linetrip”: 0 means no line trip after this fault). Note that the presence of line trip has no impact on the fault type with the generator. Moreover, the user can specify multiple faults, e.g., to define the second fault, one can use the “id”: 2 after the curly bracket] of the first fault definition and define the value of attributes. Please see [5] for more details.

D. INTEGRATION WITH OpenDSS
RAPID also utilizes the json module to define the distribution network to be coupled with the transmission network within the Parareal framework. In the dist.json file, the user can activate the T&D co-simulation (e.g., “active”: 1) and chooses a substation to include detailed dynamic models of the distribution network (e.g., “index”: [2] means the detailed representation of the distribution network at bus 2). The T&D co-simulation can be run by adding the following argument to the command line (1):

```
-dist dist.json
```

Note that the current version of RAPID only allow a single distribution network for each T&D co-simulation run. The next version will improve its features to include multiple distribution networks with different types of distributed generator models (including user-defined models).

E. OUTPUT FILE
To facilitate easy analysis of simulation outcomes, solutions are formatted using the csv form in the result.csv file. With this way, one can directly utilize the built-in functions in Excel and/or read the result.csv file in other programming languages (e.g., MATLAB) to better utilize a great variety of functions for graphical representation of data. In the result.csv file, the first column corresponds to the simulation time and other columns correspond to the solution values of state and algebraic variables. Each row corresponds to each time step. Here, the value of N is equivalent to the number of processors specified in the command line. The user can also specify the index for the variable of interest and save only those variables in the output file with the idxiwant inside the para_real.py file. The default is to only save generator rotor angles using idxiwant = idxdelta.

```
\[
\begin{bmatrix}
\text{Col}1 & \text{Col}2 & \cdots & \text{Col}M \\
\text{t}_0 & \text{Sol}_1(\text{t}_0) & \cdots & \text{Sol}_{M-1}(\text{t}_0) \\
\text{t}_1 & \text{Sol}_1(\text{t}_1) & \cdots & \text{Sol}_{M-1}(\text{t}_1) \\
\vdots & \vdots & \ddots & \vdots \\
\text{t}_N & \text{Sol}_1(\text{t}_N) & \cdots & \text{Sol}_{M-1}(\text{t}_N)
\end{bmatrix}
\]
```

IV. TIME-DOMAIN SIMULATION
A. POWER SYSTEM MODEL
The power systems are in general represented by a set of nonlinear DAEs. Thus, the simulation of electro-mechanical transient and dynamic behavior of power systems involves the computation of large DAE system using numerical time-domain simulation methods. The form of such DAE system may be represented as:

\[\dot{x} = f(x, y)\] (2a)
\[0 = g(x, y), \quad x(0) = x_0\] (2b)

where \(x\) and \(y\) represent the vectors of state variables and other algebraic, non-state variables, respectively. The differential equations (2a) include the models of synchronous generators, the associated control systems (e.g., excitation and turbine-governor systems), dynamic loads, and other dynamic devices. The algebraic equations (2b) include the static loads (e.g., the ZIP load models) and the power balance equations (the network equations) in the transmission network.

B. T&D CO-SIMULATION
As stated above, the increased dynamics and uncertainty in modern power systems are already caused in significant part from increasingly active distribution systems at the grid edges. Distributed energy resources interact with the grid and can have significant impact on the dynamics of the overall system. Any system model that aims at capturing this interplay inevitably needs to include detailed dynamic models of distribution systems. This raises the following problems.
First, transmission system simulation is performed under the assumption of balanced conditions using single-phase positive-sequence models. One exception to this is unbalanced faults which also include negative- and, perhaps, zero-sequence models. Distribution system analysis, on the other hand, is performed assuming unbalanced conditions and three-phase models. The second problem arises when these two different models are coupled together. The size of the coupled system may increase several orders of magnitudes as there can be 10s of distribution buses for any one transmission bus, each with 100s to 1000s of individual loads and distributed resources. Hence, this motivates the use of parallelization techniques for the solution of the distribution systems in the co-simulation.

In the spirit of RAPID, the goal is to solve multiple distribution systems in parallel with the transmission system simultaneously. Furthermore, the solution for each distribution system can also be performed in parallel for yet another level of parallelism with an appropriate tool. One such tool is OpenDSS, a widely-used, utility-grade distribution system simulator [40]. OpenDSS has several simulation modes available including Dynamic mode used for electro-mechanical transients. As of time of writing the paper, the level of parallelism depends on the computational platform. Under Windows, a single instance of OpenDSS can run one or more distribution systems in parallel. Under Linux, running multiple distribution systems in parallel is not inherently available due to a lag in development. As mentioned before, the current version of RAPID only allow a single distribution network for each T&D co-simulation run. We will extend this capability in the future as new libraries become available. If desired, the user can employ explicit parallel programming techniques so that multiple distribution systems can be run on multiple instances of OpenDSS on different CPU cores.

When the T&D co-simulation is implemented using separately modeled transmission and distribution systems, a general method for iterative coupling of the two models in case of a static computation (i.e., power flow) is illustrated in Fig. 2. The T&D co-simulation starts with assumed initial complex voltages in the transmission system, either a flat start at the very beginning or a previous solution. After one iteration in the transmission system, the voltages at the connecting buses are used as source voltages for one iteration in the corresponding distribution systems. Typically, only a few co-simulation iterations are required to achieve convergence.

The same slightly modified scheme can also be implemented in dynamic simulations. Since the dynamic models of standard synchronous generators are interfaced with the network using Norton equivalents, the same can be done with active distribution systems at their points of interconnection. The computed source currents from the distribution simulator represent updated injections of the equivalent Norton sources which are passed on to the main solver in each step.

V. SOLUTION APPROACH AND ALGORITHM

A. PARAREAL ALGORITHM

The main parallelization technique employed in RAPID is the Parareal algorithm which corresponds to the class of parallel methods for temporal discretization. In a simplified way, the Parareal algorithm decomposes the whole simulation period into smaller time intervals such that the evolution of each independent sub-interval is carried out in parallel and completed after a number of iterations between a coarse approximate solution and a fine true solution over the whole period. In other word, the Parareal algorithm computes the numerical solution for multiple sub-intervals simultaneously. Its graphical illustration is represented in Fig. 3.

![FIGURE 2. General structure of a T&D co-simulation.](image)

![FIGURE 3. Graphical illustration of the Parareal algorithm.](image)
As shown, to solve the sub-intervals independently, initial states for all sub-intervals are required and provided by a less accurate but computationally cheap numerical integration method (i.e., the coarse operator). The fine operator, which is more accurate but computationally expensive numerical integration method, is then performed to find the true trajectory and correct the evolution of each independent sub-interval in parallel. This process iterates until all the coarse solutions converge to the true solution. Note that the Parareal algorithm should theoretically converge to the fine solution (i.e., true solution) in $N$ iterations, which is equivalent to solving DAEs using the fine operator sequentially for all $N$ intervals.

For the Parareal algorithm, RAPID has utilized the distributed Parareal algorithm [41]. It improves upon the computational performance of the regular Parareal algorithm by also distributing the coarse propagation across all processors and thus enabling direct overlap between the sequential and parallel portions. With this, the memory requirements are uniformly distributed among processors.

### B. SEMI-ANALYTICAL SOLUTION METHOD

Traditional numerical solvers are inefficient and time-consuming in simulating real-life interconnected power grids. When a numerical integration method is used to obtain a convergent and accurate solution, a small integration time step is usually required, e.g., 1 millisecond, which requires a few minutes or even longer to complete a simulation run. Another concern with explicit integration methods such as the RK4 methods, which are widely applied in commercial simulation software, is numerical instability. Hence, a majority of utility companies still conduct power system simulation offline. The mechanism of iterative computation with numerical integration methods is a key cause of power system simulation being time-consuming for utility-scale system models.

Intuitively, if the initial value problem (2) could be solved analytically, each state variable can be represented as a closed-form analytical function about symbolic variables such as time ($t$), initial values of state variables ($x_0$), and other parameters on the operating condition ($\mu$), which may be represented as:

$$x(x_0, \mu, t) = \sum_{n=0}^{\infty} x_n(x_0, \mu, t)$$

$$\approx \sum_{n=0}^{m} x_n(x_0, \mu, t) = x_{SAS}(x_0, \mu, t)$$

where $m$ is the order of the SAS method.

Theoretically speaking, the exact solution in a closed analytical form for each state variable does not exist in general for a set of nonlinear power system DAEs. However, it is still possible to find an approximate closed-form analytical solution for each state variable, e.g., a decomposed form of the exact solution as an infinite series of simple analytical terms converging to the solution as shown in (3). And, an explicit and analytical expression equal to the sum of a finite number of terms from that infinite series may match well the exact solution for a certain time interval, say $t_1$ seconds as illustrated in Fig. 4. Thus, producing a simulated trajectory over the desired simulation window becomes evaluating the same analytical expression repeatedly for a sequence of time intervals of $T$ until making up the simulation window. Meanwhile, the initial state for each interval will take the state calculated at the end of the previous interval. Based on such an idea, a new SAS approach has been proposed in recent years as a better fit for parallel power system simulation using high-performance supercomputers.

The basic idea of SAS methods is to decompose the resolution of complex DAEs into computations at two stages: In Stage 1, whose tasks can be conducted ahead of simulation, the approach mathematically derives a high-order, approximate, and analytical solution (i.e., SAS), which retains accuracy for a certain time interval; in Stage 2, the approach plugs values into symbolic variables of the analytical solution over consecutive intervals until making up the desired simulation period. Since the computation burdens, including derivation of the SAS, can be shifted to Stage 1, simulation can focus on Stage 2 in an extremely fast manner without numerical computation.

Note that there can be multiple ways to derive such unknown coefficients $x_1, x_2, \ldots, x_m$ in (3). As discussed before, RAPID implements two promising time-power series-based SAS methods (i.e., MADM and MHAM). The user may see derived SAS terms in `en.py`. The default value of $m$ is 10 for MADM and 3 for the MHAM. The MHAM has one more adjustable parameter $c$ (referred to as the convergence-control parameter), and its default value is $-1$.

### C. ADAPTIVE MODEL REDUCTION

The AMR method adopted in RAPID was proposed in [20], [21], and [22], which divides a power system into two areas: 1) the study area, which is the main interest of an investigator, where all details are preserved, and all disturbances are originated from; 2) the external area, which can be simplified and reduced. The partitioned power system is shown in Fig. 5. The study area of the original system is connected to the external area by several tie-lines. In the external area, only the generators highly affected by the disturbance are described by nonlinear functions while the rest are linearized. The determination on whether the model of a generator is linearized or not is based on its electrical distance from the study area calculated by the column norms of the system’s admittance matrix. The resulting hybrid model of the external area can...
be represented by:

\[
\begin{align*}
\dot{\tilde{x}} &= T \left( \tilde{f}(\tilde{T}\tilde{x}, y) \right) \\
0 &= g(\tilde{T}\tilde{x}, y)
\end{align*}
\]

where \( T \) matrix transforms the original state space to the new state space on \( \tilde{x} \) with a reduced dimension by using the balanced truncation method [42], \( T \) inverts that transform back to the original state space, \( \tilde{f} \) is the set of all functions that are kept nonlinear in the hybrid model and \( f_l \) is the linearization on the rest of functions in the model. Since not all disturbances originated in the study area energize significantly nonlinear dynamics in the external area, the AMR algorithm switches the model of the external area among three models of different complexities according to the variation in system states as illustrated by Fig. 6 from [20]. The first model is the original model used for fault-on simulation, the second model is the reduced hybrid model in (4) used for a majority of the simulation period, and the third model is the linearized external system model used only when monitored post-fault state variables (e.g., the rotor angle or speed) of all generators become smaller than a preset threshold. To address any large variation of the operating condition during the simulation, linearized functions in the hybrid model can be updated.

Simulation with two areas is conducted iteratively as follows. For the study area, a tie-line \( j \) is treated as a simple fictitious generator with the internal voltage phasor equal to a voltage phasor \( V_j^{\theta_j} \) of the corresponding boundary bus in the external area and with the armature resistance and transient reactance equal to the resistance and reactance of a tie-line \( j \). Likewise for the external area, a tie-line \( j \) is treated as a generator with the internal voltage phasor equal to a voltage phasor \( V_j^{\theta_j} \) of the corresponding boundary bus in the study area. During each iteration of the system simulation these fictitious generators are treated as constant voltage sources and represent the current injections from one area to the other area. Therefore, voltage magnitudes and voltage angles of boundary buses in one area are the inputs to the model of the other area. At every iteration, each area is calculated separately, then the boundary bus voltage phasors of both areas are recalculated, and their values are sent as inputs to the corresponding area to perform the next iteration.

As constant voltage sources, the fictitious generators do not have inertias or contribute to the dynamics of each area as a component of the differential equations of generators inside the corresponding area.

**VI. CASE STUDIES**

**A. SIMULATION SETUP AND DYNAMIC MODELS**

This section presents numerical case studies to illustrate several features of RAPID. In our experiments, two test networks are considered; the New England 10-generator 39-bus system [43] and the Polish 327-generator 2383-bus system [8] that are widely used as representative test networks for the small size and large size power system network, respectively. A wide range of fault scenarios is considered: 1) 3-phase faults on buses and branches with 4 to 6 cycles fault duration; 2) 3-phase faults on branches followed by tripping of that branch with 4 to 6 cycles fault duration; 3) generator trips. Note that this section only provides the representative result with the 3-phase faults with 4 cycles using the Polish system. For all fault scenarios, the 10s simulation is conducted.

For the power system model in (2), we employ two different synchronous generator models: dummy coil model [43] and 6th order generator model [44]. Both are based on IEEE Model 2.2 [45] with two damper windings on the q-axis and one damper winding on the d-axis along with the field winding for the synchronous generators, IEEE Type 1 excitation system, and first order turbine-governor models. Loads are modelled as aggregate static loads employing polynomial representation (ZIP load). The adoption of two different
models is to demonstrate the flexibility of RAPID for modifying the codes and adding user defined functions, which implies that the user can consider different dynamic models of interest. The current version of RAPID only incorporates the dummy coil model for the synchronous generators, and its next version will include a variety of different synchronous generator models.

B. PERFORMANCE OF PARAREAL ALGORITHM

For the dummy coil model [43], RAPID incorporates four different coarse operators: the Trap method; MADM; MHAM; and AMR. For the fine operator, the RK4 method is employed. Notice that the user can switch over to a different coarse operator in Coarse.py. If desired, the user also can consider a different solution method for the fine operator in Fine.py. For the 6th order generator model, this paper incorporates three different coarse operators [19], [46]: the simultaneous-implicit method based on the trapezoidal rule (SI-Trap); the modified Euler (ME) method, and differential transformation (DT) method (another SAS method). For the fine operator, the SI-Trap method is employed. The value of 0.01 is used for the tolerance of Parareal convergence for all scenarios.

1) CONVERGENCE

This section first validates the convergence of Parareal algorithm with different coarse operators. Fig. 7 shows simulation results of Parareal algorithm for the dummy coil model of the 10s simulation with the Polish system. These results are obtained from 50 processors for parallel computing of the fine solver. In each processor, 40 sub-intervals are used for the coarse operators, and 1000 sub-intervals are used for the fine operator. The true solution is obtained using the SI-Trap method with the time step of 0.001s. Similarly, the Parareal solution converged to the true solution with all coarse operators considered for all other variables and disturbances.

2) APPROXIMATE SPEEDUP

This section approximately evaluates the computational efficiency of Parareal algorithm by considering the approximate speedup with the assumption of negligible communication time; note that results in this section are based on tests on a personal computer with the limited number of processors (e.g., 50 processors) and approximate CPU time for each operator. With this setup, the approximate runtime ($t_{\text{approx}}$), in the distributed Parareal algorithm, can be given as:

$$t_{\text{approx}} = T_c + \frac{k}{M}(T_f + T_c)$$

where $M$ and $k$ represent the number of processors (e.g., 50 for the dummy coil model and 10 for 6th order generator model) and the number of Parareal iterations for convergence, respectively; $T_c$ and $T_f$ refer to the coarse and fine propagation times, respectively over the whole simulation time period.

In this section, the number of coarse intervals for the dummy coil model is increased from 10 to 45 to reduce the number of Parareal iterations. Other settings have remained unchanged as in the previous section. Table 1 shows the approximate CPU times in seconds taken by 10s simulation and approximate speedup with the Parareal algorithm. Notice that the values of $T_c$ and $T_f$ are different in each Parareal iteration, but their variations are very small. Hence, the average value is used, and such small variations are neglected. Note that $T_f$ is identical for all methods since all methods use the same fine operator.

For the dummy coil model, the value of $T_c$ for the Trap method is higher than others since the AMR method employs the approximate model, and the SAS methods avoid the
TABLE 1. Approximate CPU times (s) and speedup with the Parareal algorithm.

| Dummy Coil Model | Method          | Trap | AMR       | MADM (m = 5) | MHAM (ε = 0.9) |
|------------------|----------------|------|-----------|--------------|----------------|
| Fine time (T_f)  | 72.63          | 77.63| 77.63     | 77.63        |                |
| Coarse time (T_c)| 15.95          | 11.31| 12.17     | 11.62        |                |
| # of iterations  | 4              | 4    | 4         | 4            |                |
| Parareal time (T_{p}^{par}) | 21.28 | 18.45| 19.35     | 18.76        |                |
| Fine alone time  | 77.63          | 77.63| 77.63     | 77.63        |                |
| Speed Up         | 3.64           | 4.21 | 4.01      | 4.14         |                |

TABLE 2. The full EI simulation result (s) with HPC.

| Processors | Last Processor | Fine | Coarse | Communication | Total Time |
|------------|----------------|------|--------|---------------|------------|
| 10         | 857            | 774  | 40.3   | 42.7          | 857        |
| 100        | 243            | 125  | 7.24   | 56.4          | 243        |
| 250        | 178            | 73.6 | 4.67   | 93.5          | 179        |
| 500        | 145            | 46.1 | 3.33   | 85.3          | 145        |

numerical iteration. Note that the acquired speedup from all coarse operators in this case is about 4 considering the 4 iterations. Our experience from multiple experiments suggests the followings: 1) the Parareal algorithm showed more stable convergence with the SAS methods, 2) the Trap method typically requires less iterations for Parareal convergence than other methods, 3) the AMR method as the coarse operator is faster than others but it may require more iterations in some cases than other methods, and 4) the windowing approach [13], which runs a smaller time window (e.g., 1s simulation) multiple times sequentially for a longer time simulation (e.g., 10s simulation), can significantly improves the Parareal algorithm.

For the 6th order generator model, the acquired speedup shows variations for each coarse operator. In particular, the speedup from the DT method is about 9 due to the lower value of T_c and 1 iteration for convergence indicating its efficient computation and high accuracy. The approximate speedup results for the 6th order generator model come from the solution methods all based on the simultaneous-implicit method whose fundamental solution procedure is different from the explicit method used in the dummy coil model. The average speedup from various experiments considering more than 100 different cases was about 8, and for some cases, the speedup was more than 10. Note that the dummy coil model and 6th order generator model use a different solution procedure. For the dummy coil model, SAS methods are only applicable to solving ODEs and yield the same time for solving algebraic equations, which results in relatively smaller differences in T_c. For the 6th order generator model, different methods are applied to the entire DAEs and thus can yield a significantly different value of the T_c in each method.

C. RESULTS WITH THE HPC PLATFORM

To further verify the approximate speedup described in the previous section and practical values of Parareal algorithm, we have deployed RAPID on Compute and Data Environment for Science (CADES) HPC computing platform [47] for conducting the simulations and profiled the simulations from serial to 100’s of processes. The CADES condo machine consists of 160 nodes with 32 core per node with 124 GB RAM.

This test conducts the 10s simulation with the dummy coil model and considers the 3-phase fault at bus 1 with 4 cycles using the Eastern Interconnection (EI), 5,617-generator and 70,285-bus system [48], which has been tested and officially validated by the EI reliability assessment group. Also note that this test does not simplify DAEs for the EI system, i.e., it uses the same dynamic model as used in the previous sections. For the Parareal algorithm, the Trap method and RK4 method are used as the coarse operator and the fine operator, respectively. The time steps of 0.02s and 0.002s are used for the coarse operator and fine operator, respectively in each case. Fig. 9 shows the rotor angle of one of generators in the EI system and its convergence to the true solution.

Considering the different number of processors, the resulting computational time (sec) of Parareal algorithm using the HPC platform is shown in Table 2. As the number of processors increases, the computational performance is significantly improved; the Parareal algorithm with 500 processors is completed about 6 times faster than one with 10 processors. This is because as we scale the problem to large processors, the algorithm allows one to have more (smaller) subintervals and thus more parallel computations in the fine and coarse operators, which leads to reducing the computation time in each operator on each processor significantly. While the simulation with more processors increases the communication time, the reduction from a larger number of parallel computations is more significant and thus reduces the total simulation time.

These results demonstrate the great potential of Parareal algorithm equipped with emerging solution techniques to achieve an ambitious goal of facilitating “faster than real-time simulation” for predicting power system dynamic behaviors. Along with this initial evidence, a more extensive
study using the HPC system is an important future research direction to fully demonstrate its potential.

D. RESULTS WITH T&D CO-SIMULATION

To illustrate the T&D dynamic co-simulation framework, a large realistic distribution system (6058 buses with 52.12 MW and 11.63 Mvar), EPRI Circuit 24 shown in Fig. 10, has been modeled in OpenDSS. This system is originally passive and, to make it dynamic, an inverter-interfaced distributed generator (IIDG) has been connected to the internal bus adjacent to the source. Generator Model 7 is used as an IIDG with power equal to the total load in the system for 100% penetration. This generator model approximates a simple inverter. It generates constant power, but limits the current if the voltage falls below a specified value. The whole distribution system has been connected to bus 8 of the New England system used for the transmission part, shown in Fig. 11.

The T&D co-simulation test conducts the 10s simulation with the dummy coil model. For the Parareal algorithm, the Trap method and RK4 method are used as the coarse operator and the fine operator, respectively. The time steps of 0.02s and 0.002s are used for the coarse operator and fine operator, respectively. A temporary symmetrical fault was created at the interconnection between the two systems, and the voltage and the injected current fluctuation at that bus impacted by the distribution network are shown in Fig. 12, which demonstrates the successful implementation of the dynamic T&D co-simulation using Parareal algorithm. Regarding the computational performance, our experience with the New England system showed that the T&D co-simulation with one distribution system takes about twice as much time to finish the simulation for each operator (i.e., both fine and coarse operators) as compared to the simulation time without the distribution system. Thus, the current Parareal algorithm implementation with one distribution system takes about twice as much time as compared to the Parareal algorithm without the distribution system. The additional computational burden also depends on the simulation setup and scenarios (e.g., fault types) and may further increase if one considers multiple distribution systems without running them in parallel.

E. DISCUSSION

As previously noted, the adoption of HPC computing platforms as well as other emerging solution techniques for power system simulations is of great interest due to the significant improvement on the computational performance of the time-domain simulation. To the best of our knowledge, RAPID is the first attempt to develop a Python-based open source power system simulation tool that incorporates such emerging solution techniques. Our observations from numerical case studies imply that RAPID has a great potential to promote and accelerate R&D in modeling and simulation techniques for power systems. We provide several key summaries for RAPID:

- RAPID is a Python-based open source tool to solve large-scale nonlinear power system DAEs. To improve the computational performance, RAPID incorporates emerging solution techniques (i.e., parallel-in-time algorithm, SAS methods, and AMR techniques). Therefore, RAPID as a highly flexible open source tool would allow other interested researchers to easily understand such techniques, test those with their system, and further advance those for more novel computational frameworks. Also, RAPID provides the T&D co-simulation framework.
VII. CONCLUSION

This paper has presented a new Python-based open-source tool RAPID for large-scale power system dynamic simulations. RAPID incorporates the emerging solution techniques to improve the computational performance of the time-domain simulation and provides a high-level set of such techniques for researchers as well as educators and students. Therefore, it has a great potential to mitigate challenges associated with exploring new techniques outside of power system for solving a large number of nonlinear power system DAEs and thus promote and advance research in dynamic modeling and simulation techniques as well as developing new control and protection schemes. Currently, RAPID is actively maintained by the authors, and future projects will continuously upgrade its features by adding new components and solution techniques as well as making it more user-friendly, general and flexible.

ACKNOWLEDGMENT

The authors wish to thank the editor and the anonymous reviewers for providing valuable feedbacks on this work. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, and world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes. The Department of Energy (DOE) will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

REFERENCES

[1] A. Bose and T. J. Overbye, “Electricity transmission system research and development: Grid operations,” in Proc. Transmiss. Innov. Symp., Modernizing U.S. Electr. Grid, Apr. 2021, pp. 1–63.

[2] A. J. Flueck, “High-fidelity, faster than real-time dynamics simulation,” in Proc. IEEE PES Gen. Meeting | Conf. Expo., Jul. 2014, p. 1.

[3] “1,200 mw fault induced solar photovoltaic resource interruption disturbances report,” North Amer. Electr. Rel. Corp., Atlanta, GA, USA, Tech. Rep., Jun. 2017.

[4] “April and May 2018 fault induced solar photovoltaic resource interruption disturbances report,” North Amer. Electr. Rel. Corp., Atlanta, GA, USA, Tech. Rep., Jan. 2019.

[5] S. Simunovic, S. Allu, B. Park, K. Sun, and A. Dimitrovski. (Sep. 2021). Resilient Adaptive Parallel Simulator for Grid (Rapid): Computer Software. [Online]. Available: https://code.ornl.gov/PowerSystems/rapid

[6] F. Milano, “An open source power system analysis toolbox,” IEEE Trans. Power Syst., vol. 20, no. 3, pp. 1199–1206, Aug. 2005.

[7] J. H. Chow and K. W. Cheung, “A toolbox for power system dynamics and control engineering education and research,” IEEE Trans. Power Syst., vol. 7, no. 4, pp. 1559–1564, Nov. 1992.

[8] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, “MATPOWER: Steady-state operations, planning and analysis tools for power systems research and education,” IEEE Trans. Power Syst., vol. 26, no. 1, pp. 12–19, Feb. 2011.

[9] L. Thurner et al., “Pandapower—An open-source Python tool for convenient modeling, analysis, and optimization of electric power systems,” IEEE Trans. Power Syst., vol. 33, no. 6, pp. 6510–6521, Nov. 2018.

[10] H. Cui, F. Li, and K. Tomsovic, “Hybrid symbolic-numeric framework for power system modeling and analysis,” IEEE Trans. Power Syst., vol. 36, no. 2, pp. 1373–1384, Mar. 2021.

[11] J. Lions, Y. Maday, and G. Turinici, “A parareal in time discretization of PDE’s,” Comptes Rendus de l’Academie des Sci. Ser. I Math., vol. 332, no. 7, pp. 661–668, Apr. 2001.

[12] 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.

[13] 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., vol. 36, no. 6, pp. 5068–5080, Nov. 2021.

[14] T. Cheng, T. Duan, and V. Dinavahi, “Parallel-in-time object-oriented electromagnetic transient simulation of power systems,” IEEE Open Access J. Power Energy, vol. 7, pp. 296–306, 2020.

[15] W. Wang, N. Duan, and K. Sun, “A time–power series-based semi-analytical approach for power system simulation,” IEEE Trans. Power Syst., vol. 34, no. 2, pp. 841–851, Mar. 2019.

[16] D. L. Dinesha and G. Gurrala, “Application of multi-stage homotopy analysis method for power system dynamic simulations,” IEEE Trans. Power Syst., vol. 34, no. 3, pp. 2251–2260, May 2019.

[17] N. Duan and K. Sun, “Application of the adomian decomposition method for semi-analytic solutions of power system differential algebraic equations,” in Proc. IEEE Eindhoven PowerTech, Jun. 2015, pp. 1–6.

[18] G. Gurrala, D. L. Dinesha, A. Dimitrovski, P. Sreekanth, S. Simunovic, and M. Starke, “Large multi-machine power system simulations using multi-stage Adomian decomposition,” IEEE Trans. Power Syst., vol. 32, no. 5, pp. 3594–3606, Sep. 2017.

[19] Y. Liu and K. Sun, “Solving power system differential algebraic equations using differential transformation,” IEEE Trans. Power Syst., vol. 35, no. 3, pp. 2289–2299, May 2020.

[20] D. Osipov and K. Sun, “Adaptive nonlinear model reduction for fast power system simulation,” IEEE Trans. Power Syst., vol. 33, no. 6, pp. 6746–6754, Nov. 2018.

[21] D. Osipov, N. Duan, A. Dimitrovski, S. Allu, S. Simunovic, and K. Sun, “Adaptive model reduction for parareal in time method for transient stability simulations,” in Proc. IEEE Power Energy Soc. Gen. Meeting (PESGM), Aug. 2018, pp. 1–5.

[22] 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,” in Proc. IEEE Power Energy Soc. Gen. Meeting (PESGM), Aug. 2019, pp. 1–5.

[23] B. Park et al., “Performance and feature improvements in parareal-based power system dynamic simulation,” in Proc. IEEE Int. Conf. Power Syst. Technol. (POWERCON), Sep. 2020, pp. 1–6.
SRIKANTH ALLU received the Ph.D. degree in mechanical engineering from the University of Kansas, Kansas, USA, in 2008. He is a Computational Scientist with the Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA.

BYUNGKWON PARK (Member, IEEE) received the B.S. degree in automation and the Ph.D. degree in control science and engineering from Tsinghua University, Beijing, China, in 1999 and 2004, respectively. He was the Project Manager in grid operations and planning with the Electric Power Research Institute (EPRI), Palo Alto, CA, USA, from 2007 to 2012. He is a Professor with the Department of EECS, The University of Tennessee Knoxville, Knoxville, USA. He is an Associate Editor of IEEE TRANSACTIONS ON POWER SYSTEMS and IEEE OPEN ACCESS JOURNAL OF POWER AND ENERGY.

ALEKSANDAR DIMITROVSKI (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees in electrical engineering with emphasis in power and the M.Sc. degree in applied computer sciences in Europe. He is an Associate Professor with the Electrical and Computer Engineering Department, University of Central Florida (UCF). Before joining UCF, he had worked in research institutions, power industry, and academia both in the USA and Europe. His research interests are focused on modeling, analysis, protection, and control of uncertain power systems.

YANG LIU (Member, IEEE) received the B.S. degree in energy and power engineering from Xi’an Jiaotong University, China, in 2013, the M.S. degree in power engineering from Tsinghua University, China, in 2016, and the Ph.D. degree in electrical engineering from The University of Tennessee Knoxville, Knoxville, USA, in 2021. He is currently a Post-Doctoral Researcher with The University of Tennessee Knoxville. His research interests include power systems simulation, dynamics, stability, and control.

SRDJAN SIMUNOVIC received the M.Sc. and Ph.D. degrees in civil engineering from Carnegie Mellon University, Pittsburgh, PA, USA. He is a Distinguished Research Staff with the Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA. His research expertise include applied mathematics for multi-scale and multi-physics problems, parallel algorithms and computing, energy storage, and computational modeling of materials and processes.