RE-INTEGRATE EMT Simulation Software: DAE Solvers and Automation

Phani R V Marthi[1] marthip@ornl.gov
Soumyajit Gangopadhyay[1,2] szg6052@psu.edu
Kuan-Chieh Hsu[3] khsu037@ucr.edu
Jongchan Choi[1] choij1@ornl.gov
Nilanjan Ray Chaudhuri[2] nuc88@psu.edu

Suman Debnath[1]
debnaths@ornl.gov

***Abstract*—Existing electromagnetic transient (EMT) simulation tools face challenges in accelerating EMT simulations, especially for very large-scale power networks. To tackle this issue, next generation EMT simulation tools such as RE-INTEGRATE EMT are being researched upon. Such tools should be equipped with automation capabilities and advanced numerical differential-algebraic equation (DAE) solvers. In this paper, the DAE solvers incorporated within the RE-INTEGRATE EMT simulation tool are discussed. In particular, a modified ODEINT-based DAE solver and the ARKODE solver from SUNDIALS are leveraged within RE-INTEGRATE EMT. In addition, the automation implemented within RE-INTEGRATE EMT to automate the DAE generation (replacing the need of manual discretization and assembling DAEs) is discussed. Different use cases were implemented using the RE-INTEGRATE EMT tool and were validated with respect to baseline simulations.**

***Index Terms*—Electromagnetic Transient, EMT, differential-algebraic equations, DAEs, stiff solvers, non-stiff solvers**

I. Introduction

With the increasing integration of power electronics systems and smaller electrical machines into the power grid, faster timescale events such as transient events, oscillations, and control interactions are observed more frequently. Electromagnetic transient (EMT) simulation tools are needed to accurately capture these events [1] since such tools can simulate the electromagnetic behavior of electrical systems that are observed during fast changes spanning a few microseconds or smaller.

Different EMT simulation tools are available today such as PSCAD, EMTP, among others. These tools are based on the nodal approach [2]. However, the major challenges associated with EMT simulation of a large-scale power system equipped with power electronics based systems such as inverter-based resources (IBRs), inverter-based loads, and distributed energy resources (DERs) include: (a) limited scalability of power grid simulation; (b) lack of automation; and (c) speed-up limited by computational constraints. To address these challenges, newer generation of EMT simulation tools have to be researched upon. These newer generation of EMT simulation tools should be infused with state-of-the-art advanced numerical solvers, automation algorithms, high-performance computing (HPC) techniques, and artificial intelligence algorithms.

Different from the existing nodal analysis based EMT simulation tools, this work presents a new software framework for EMT simulations that generates and discretizes the differential-algebraic equations (DAEs) of a power system. This newer generation EMT simulation tool, called RE-INTEGRATE, includes DAE solvers and automation pipelines required to accelerate EMT simulations. In this paper, the DAE solvers that are currently utilized within RE-INTEGRATE are discussed. One of the DAE solvers is developed by modifying ODEINT [3], a library that was originally designed for solving ordinary differential equations (ODEs). The other solver is the Implicit-Explicit (ImEx) solver adopted from the Suite of Nonlinear and Differential/Algebraic Equation Solvers (SUNDIALS) [4], [5]. This paper also outlines the automation processes currently implemented within RE-INTEGRATE in order to automate (a) the process of DAE generation of an electrical network and (b) the process of discretization involved in solving the DAEs. Two different use cases that were implemented using the DAE solvers are documented that confirms the accuracy of the developed EMT software.

II. RE-INTEGRATE EMT Simulation Tool

RE-INTEGRATE EMT simulation tool is the next generation EMT simulation tool that can perform EMT simulation of large-scale power grids with power electronics (PE)-based systems and other fast-acting components. This section describes the main pillars of RE-INTEGRATE EMT that include automation, DAE solvers, linear solvers, and intelligence. Other pillars of this simulation tool such as models/libraries and features extraction will be described in future papers.

A. Automation

Automation consists of five sequential steps. At first, a parser is developed to process a netlist (see Fig. 4b for an example) that characterizes the system under study. For each component in the netlist, the parser first creates an object of the component's class and then initializes it with the component's parameters read from the netlist[1]. Based on the source and the

Research sponsored by Applied Grid Modeling (AGM) program in U. S. Department of Energy's Office of Electricity (OE). The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government. [1]Energy Science & Technology Science Directorate, Oak Ridge National Laboratory, Knoxville, TN, USA. [2]Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA, USA. [3]Department of Computer Science, University of California Riverside, Riverside, CA. Currently he is with Brookhaven National Laboratory. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. 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, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (https://www.energy.gov/doe-public-access-plan).

[1]Currently, the parser can process resistance, inductance, capacitance, voltage source, and current source only. This is being extended to other power system components like generators, exciters, power system stabilizers, etc.

destination nodes of the components stored in the components' objects, an undirected graph is then formed where each node serves as a vertex and a component connected between two nodes serves as an edge of the graph. Since loop identification enables KVL analysis, identifying the components forming a loop in the connectivity graph becomes necessary to obtain a network's DAEs. To this end, a modified Depth First Search (DFS) algorithm is developed to identify the loops such that no two loops lead to the same DAE upon KVL analysis. Post loop identification, KVL analysis is conducted on each of the m identified loops, resulting in m DAEs. Then, KCL analysis is conducted on $n-1$ nodes for a network with n nodes, resulting in $n-1$ DAEs. Consequently, for a system with n nodes and m loops, $m+n-1$ DAEs are formed. Among these $m+n-1$ DAEs, there may exist trivial DAEs discussed further in section III-A. At first, such DAEs (if any) are eliminated. Then, the remaining non-trivial DAEs are separated into the ODEs and the algebraic equations (AEs).

B. DAE Solvers

The RE-INTEGRATE EMT tool employs DAE solvers that are developed by modifying and incorporating the ODEINT and the SUNDIALS libraries. Consequently, RE-INTEGRATE EMT houses different DAE solvers having different orders of discretization, different stiffness-based discretization methods, among others, that can facilitate dynamic simulation of systems defined by DAEs with different properties. The DAE solvers are automatically invoked at the end of the automation step in order to discretize the generated DAEs, leading to a linear system of equations at each time step during the simulation interval obtained as an input from the user. More details on the DAE solvers are available in section III.

C. Linear Solvers

Based on the discretization of the DAEs representing EMT simulation dynamic models, a linear system of equations having the form $\mathrm{Ax} = \mathrm{b}$ (where A is a square matrix) is generated and solved at each time step of the simulation. For a large-scale power system, the matrix A is not only sparse but also has special forms and properties [6]. Within RE-INTEGRATE EMT, several sparse linear solvers that can leverage these special properties and forms of the A matrix are utilized to improve the scalability and the speed of the EMT simulation of a large-scale power system. Some of the linear solvers incorporated within RE-INTEGRATE EMT include KLU [7] and SuperLU. This work does not discuss the linear solvers in detail that will be presented in future papers.

D. Intelligence

Intelligence is one of the key pillars within RE-INTEGRATE EMT. The DAEs obtained for power grids with fast-acting devices have properties that can be explored to use different types of discretization methods [8]. Moreover, the matrices resulting from large-scale power system EMT simulation models are often sparse with specific structural configurations. Intelligence is also being embedded in the RE-INTEGRATE EMT simulation tool to identify the specific structure of the matrices. These intelligent methods help in re-ordering the matrices that lend themselves to a variety of linear solvers that are appropriate for enhancing speed-up and achieving scalability in EMT simulation of large-scale power grids [6]. Intelligence is also utilized within RE-INTEGRATE for automation of large-scale systems [8].

The rest of this paper solely focuses on the DAE solvers and the automations developed within RE-INTEGRATE EMT.

III. DAE Solvers

This section outlines (a) the modifications incorporated within ODEINT and (b) the capabilities utilized from SUNDIALS to solve the DAEs generated within RE-INTEGRATE EMT.

A. ODEINT

ODEINT includes several numerical integrators (such as Euler, implicit Euler, adams-bashforth, adams-moulton, rosenbrock, and explicit runge-kutta methods) that can solve a set of ODEs, provided by the user as an input, over a pre-specified time interval set by the user. Since ODEINT can solve only ODEs, ODEINT is appropriately modified to interface it with RE-INTEGRATE EMT. In particular, ODEINT is modified to (i) automatically accept the set of DAEs generated within RE-INTEGRATE EMT as input instead of user-defined ODEs only, (ii) handle AEs in addition to ODEs, and (iii) delink the linear solvers from the DAE solvers at each time step of the simulation. The proposed modifications are as follows:

1) Automated DAE generation within RE-INTEGRATE EMT: To generate a network's DAEs, identifying the components (or edges) constituting a loop becomes necessary. This is because, for each component, the voltage-current ($v-i$) dynamics is described by a DAE that differs across components. The traditional DFS algorithm forms and stores the adjacency list of a graph that indicates whether a pair of vertices is directly connected or not. While this helps in identifying the vertices forming form a loop, the algorithm cannot identify all the loops when multiple edges connect a pair of nodes. In an electrical network, multiple components can be connected between a pair of nodes, leading to multiple loops involving the same pair of nodes. Inability to detect and identify such loops may underestimate the number of DAEs required to characterize the network. To address this issue, the DFS algorithm is modified within RE-INTEGRATE EMT. In particular, besides storing the adjacency list of the connectivity graph, RE-INTEGRATE EMT stores (a) the number of edges between any two nodes and (b) the information of the component corresponding to each edge in a form similar to that of the adjacency list. The resulting advantages are two-fold. First, multiple loops involving the same pair of nodes can be identified. Second, upon detection of a loop, the components forming the loop can be back-tracked simultaneously while back-tracking the nodes forming the loop.

It is worth noting that with the proposed modification, the DFS algorithm may identify two different loops that yield the same DAE upon KVL analysis as shown in Fig. 1a. Starting from node 0, the algorithm may identify two loops - Loop 1 and Loop 2. Since (i) a capacitor's $v-i$ dynamics is *uniquely* determined by the capacitor's voltage and (ii) C_1 and C_2 are connected in parallel, the DAEs resulting from KVL analysis of Loop 1 and Loop 2 are identical. Identical DAEs may also be encountered when two independent current sources are connected in parallel. Identical DAEs can pose singularity problems for the linear solvers[2]. To circumvent this

[2]Identical DAEs are not formed when resistors/inductors are connected in parallel since their $v-i$ dynamics are *uniquely* determined by their currents.

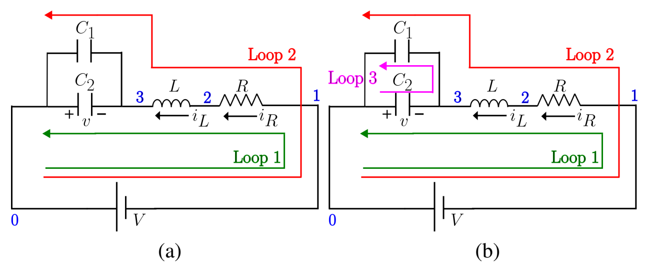


Fig. 1. (a) Identified loops: Loop 1 and Loop 2 (KVL: $-V - i_R R - L\frac{di_L}{dt} + v = 0$) (b) Identified loops: Loop 1/Loop 2 (KVL: $-V - i_R R - L\frac{di_L}{dt} + v = 0$) and Loop 3 (KVL: $-v + v = 0$).

issue, the DFS algorithm is further modified to keep track of the "reverse-parent" nodes for each node in the connectivity graph. In particular, upon a loop detection, if node X directly leads to node Y during back-tracking, then node X is assigned as a "reverse-parent" node of node Y. Instead of identifying Loop 1 and Loop 2 as the two loops, the modified algorithm identifies either Loop 1 and Loop 3 or Loop 2 and Loop 3, thus preventing the generation of identical DAEs from KVL analysis. However, as shown in Fig. 1b, KVL analysis of Loop 3 yields $v = v$ which is trivial. Such trivial DAEs are eliminated to form the final set of N non-trivial DAEs.

2) Modifying ODEINT to solve DAEs: ODEINT is capable of solving a set of ODEs of the form:

$$\dot{x_i} = f_i(\boldsymbol{x}) \qquad \forall i \in \{1, 2, \ldots, m\} \tag{1}$$

where m is a positive integer and $\boldsymbol{x} \in \mathbb{R}^m$ having coordinates $x_i \ \forall i \in \{1, \ldots, m\}$. To solve the DAEs generated within RE-INTEGRATE EMT, the implicit Euler class within ODEINT is modified to tackle linear/affine DAEs of the form (2a)-(2b).

$$\sum_{j=1}^{N} l_{ij}\dot{x_j} = \sum_{j=1}^{N} r_{ij}x_j + c_i \qquad \forall i \in \{1, 2, \ldots, N_1\} \tag{2a}$$

$$0 = \sum_{j=1}^{N} r_{ij}x_j + c_i \qquad \forall i \in \{N_1 + 1, \ldots, N\} \tag{2b}$$

where N is the total number of DAEs, out of which N_1 are ODEs and the rest are AEs. The coefficients l_{ij}, r_{ij} and c_i are real numbers $\forall i, j \in \{1, 2, \ldots, N\}$. At each time step of the simulation, the objective is to generate the matrix A_{DAE} and the vector b_{DAE} such that the solution at the current time step can be obtained by solving the linear system $A_{DAE}x = b_{DAE}$. To realize this objective, at first, the matrices J_L, $J_R \in \mathbb{R}^{N\times N}$ are formed within the modified implicit Euler class as:

$$J_L \leftarrow \begin{bmatrix} l_{11} & \ldots & l_{N_1 1} & 0 & \ldots & 0 \\ \vdots & \ldots & \vdots & \vdots & \ldots & \vdots \\ l_{1N} & \ldots & l_{N_1 N} & 0 & \ldots & 0 \end{bmatrix}^T \quad J_R \leftarrow \begin{bmatrix} r_{11} & \ldots & r_{1N} \\ \vdots & \vdots & \vdots \\ r_{N1} & \ldots & r_{NN} \end{bmatrix}$$

Then, the matrices A, B, C, D, E, F are formed as:

$$\begin{aligned} &A \leftarrow (J_L)_{i,j} \ \ i \in \{1, \ldots, N_1\}, j \in \{1, \ldots, N\} \\ &B \leftarrow (J_L)_{i,j} \ \ i \in \{N_1 + 1, \ldots, N\}, j \in \{1, \ldots, N\} \\ &C \leftarrow (J_R)_{i,j} \ \ i \in \{1, \ldots, N_1\}, j \in \{1, \ldots, N\} \\ &D \leftarrow (J_R)_{i,j} \ \ i \in \{N_1 + 1, \ldots, N\}, j \in \{1, \ldots, N\} \\ &E \leftarrow A - hC \quad \text{and} \quad F \leftarrow B - D \end{aligned}$$

where $h > 0$ is the stepsize used in the simulation. The matrix $A_{DAE} \in \mathbb{R}^{N\times N}$ and the vector $b_{DAE} \in \mathbb{R}^N$ are formed as:

$$\begin{aligned} &A_{DAE} \leftarrow \begin{bmatrix} E^T & F^T \end{bmatrix}^T \\ &b_{DAE} \leftarrow J_L x(t_n) + \begin{bmatrix} hc_1 & \ldots & hc_{N_1} & c_{N_1+1} & \ldots & c_N \end{bmatrix}^T \end{aligned}$$

where $x(t_n)$ is the solution at $t = t_n$. The solution $x(t_n + h)$ at $t = t_n + h$ is currently obtained by solving $A_{DAE}x = b_{DAE}$ using the LU solver available within ODEINT.

3) Delinking Linear solvers from DAE solvers: In the existing ODEINT codebase, the LU solver is invoked from within the implicit Euler class. Consequently, the matrix A_{DAE} and the vector b_{DAE} (generated within the modified implicit Euler class at each time step) cannot be accessed from outside the implicit Euler class. This in turn restricts the choice of the linear solver to the ones available within ODEINT. Hence, RE-INTEGRATE EMT currently delinks the linear solver routines from the DAE solver routines. As a consequence, a linear solver cannot be invoked from within the DAE solver class. Instead, A_{DAE} and b_{DAE} can now be accessed outside the DAE solver class which creates room for incorporating novel linear solvers within RE-INTEGRATE EMT in the future.

B. SUNDIALS: Implicit-Explicit Solvers

SUNDIALS has a package of ODE and DAE solvers. For this work, SUNDIALS ImEx solvers are utilized to exploit the stiffness property present within DAEs representing the dynamics of a power grid equipped with power electronics. However, SUNDIALS ImEx solvers do not incorporate first-order solvers that are available within ODEINT.

1) ImEx Solvers: Implicit-Explicit Solvers: SUNDIALS consists of a DAE solver package called ARKODE that can solve initial value problems (IVPs) given by (3).

$$x' = f_I(x, t) + f_E(x, t); \qquad x(t_0) = x_0 \tag{3}$$

where, x, t represents state of the system and time respectively; $f(x, t)$ denotes the equations that represent the dynnamics of the system. The dynamics of the system are segregated into two parts: "stiff" and "non-stiff" components. $f_I(x, t)$ contains the "stiff" components of the system and $f_E(x, t)$ contains the "non-stiff" components of the system. The ARKODE package in SUNDIALS supports one-step time implicit-explicit (ImEx) additive Runge–Kutta methods. The "stiff" components-based states in $f_I(x, t)$ are integrated using diagonally-implicit Runge-Kutta methods (DIRK), and the "non-stiff" components-based states are integrated using explicit Runge-Kutta methods (ERK) [9]. The user should provide both functions $f_I(x, t)$ and $f_E(x, t)$ for the mixed stiff/non-stiff problems that define the dynamics of the system.

2) Sources of Stiff System DAEs in EMT Simulation: The key factors that can play a key role in introducing stiffness into the DAEs are ratios of inductances, capacitances, high-frequency switching actions of power-electronics systems, rapid dynamics of various filter components, fast dynamics of the input and output voltages, and interactions between the filter components and loads. Furthermore, non-linearities in the systems also play a key role in determining the stiffness in the systems. Some examples of "stiff" system of DAEs include: arm current dynamics during sub module (SM) blocked condition in Modular multilevel converter (MMC) - High Voltage direct current (HVdc) [10]; inductor current dynamics in direct current (dc)-dc converters models, dc-alternating current (ac) inverters models, and filter currents and voltages dynamics in dc-ac inverters of the photovoltaic (PV) inverter system models, among others [11]. Some examples of the "non-stiff" system of DAEs include: SM capacitor voltage dynamics in MMC-HVdc system models [10], input and output voltage dynamics of the dc-dc converter models [11], among others.

3) ImEx Solvers: Implementation and Automation: Once the "stiff" and "non-stiff" DAEs of the system are identified, the system module is developed in C programming language using ImEx solver module ARKODE and included into $f_I(x,t)$ and $f_E(x,t)$. The "stiff" and "non-stiff" functions are then processed (discretization required for EMT simulation) using $ARKStepEvolve$ function (in the ARKODE integrator), thereby automating the entire discretization process for EMT simulation of the PV inverter module with SUNDIALS. Subsequently, the system can employ either direct or iterative solvers to solve these equations and update the state variables.

4) Linear Solvers: SUNDIALS: In this research, the matrix-based solver SUNLinSol Dense from SUNDIALS is chosen for example purposes; however, users have the flexibility to select alternative solvers from SUNDIALS or external sources.

IV. Case Studies and Simulation Results

In this section, two case studies are shown: (a) PV inverter simulation using ImEx solver in SUNDIALS; (b) Circuit simulation using the modified ODEINT based DAE solver.

A. PV inverter

The PV inverter module considered in this use-case is shown in Fig. 2. The PV inverter module consists of PV array, dc-dc boost converter, dc-ac inverter, and LCL filter. In the PV inverter EMT simulation model, the voltage and current dynamics of capacitors and inductors in dc-dc converter, dc-ac inverter, and LCL filter are represented by DAEs [11]. In [11], the EMT simulation model of the PV inverter module is simulated by state-space method. In the case of state-space approach, the PV inverter module's DAEs are assembled based on Kirchoff's current and voltage laws to form the system DAEs. The formed system DAEs are then discretized manually using hybrid discretization methods resulting in linear system of equations. The linear system of equations is then solved using generic matrix inversion methods such as LU decomposition or Gauss elimination methods.

In the proposed method using ImEx solvers, DAEs are directly included into the $f_I(x,t)$ and $f_E(x,t)$ based on the stiffness category. In the case of PV inverter module, DAEs for the input and the output voltage dynamics of the dc-dc converter are classified as "non-stiff" components and DAEs for the inductor current dynamics of the dc-dc converter, the inverter currents of the dc-ac inverter are classified as "stiff" components. These DAEs are implemented using the automation process explained in Section III-B3. The linear system of equations generated from this automated discretization process is solved at each time step using the matrix-based solver SUNLinSol Dense from SUNDIALS.

The simulation results for the PV inverter module simulation using SUNDIALS ImEx DAE solvers are shown in Figs. 3a-3c. The inductor current dynamics of the dc-dc converter, the dc-link voltage dynamics of the dc-ac inverter, and the dc-ac inverter current dynamics of the PV inverter module are shown in Figs. 3a-3c respectively (corresponding circuit components highlighted in Fig. 2). From the results, it is observed that the PV inverter module is stable, accurate and compares well with the baseline results from the existing simulators [11].

B. Circuit with resistors, inductors, capacitors and battery

RE-INTEGRATE EMT was used to simulate the circuit shown in Fig. 4a for 0.5 s. At first, the circuit was represented using the netlist shown in Fig. 4b. Then this netlist was fed to RE-INTEGRATE EMT that generated 10 DAEs (8 ODEs and 2 AEs) representing the circuit's dynamics. Within RE-INTEGRATE EMT, the implicit Euler method was set as the DAE solver and the circuit was simulated using three stepsizes: 0.1 ms, 1 ms and 10 ms. For each simulation, the corresponding stepsize was provided as an input to the DAE solver for discretization. Consequently, at each time step, the tool produced an $A_{DAE} \in \mathbb{R}^{10\times 10}$ and a $b_{DAE} \in \mathbb{R}^{10}$ which were fed to the LU solver (available within ODEINT) to solve the circuit at that time step. Note that, all six state variables (i.e., v_{0-1}, v_{5-4}, i_{2-1}, i^a_{4-3}, i^b_{4-3}, and i^c_{4-3}) were initialised to zero in the simulations. For each stepsize, the outputs obtained from RE-INTEGRATE EMT were compared with the outputs obtained by simulating the circuit of Fig. 4a in MATLAB Simulink using implicit Euler method and the same stepsize. The objective of this comparison was to verify the accuracy of RE-INTEGRATE EMT across different stepsizes. The obtained results are shown in Fig. 5. Further, the relative error ($r.e.$) between the outputs obtained from Simulink and RE-INTEGRATE EMT was evaluated as:

$$r.e.(\%) = \frac{|\text{Simulink output} - \text{RE-INTEGRATE output}|}{|\text{Simulink output}|}\times 100$$

The variation of the relative error against time is also shown in Fig. 5 for the two state variables v_{0-1} and v_{5-4}. It is seen from Fig. 5 that the outputs obtained from RE-INTEGRATE EMT are indistinguishable from the corresponding outputs obtained from Simulink. Further, the relative error is negligible in each of the six dynamics shown in Fig. 5. This demonstrates the generality of RE-INTEGRATE EMT across different stepsizes. The generality of RE-INTEGRATE EMT across different stiffness ratios was also investigated by varying the inductance (between nodes 1 & 2) and the capacitance (between nodes 4 & 5). The results obtained were found to be satisfactory. However, these results are not provided in this paper due to space limitations. Additionally, it is seen from Fig. 5 that unlike v_{5-4} which exhibits a non-oscillatory and over-damped dynamics, the state v_{0-1} exhibits a decaying oscillatory dynamics. This is because, unlike the 1.5 ohm resistance that provides significant damping, the 0.05 ohm resistance is too small to provide

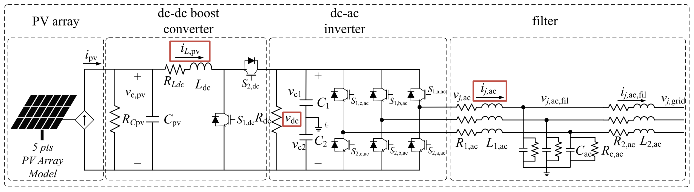
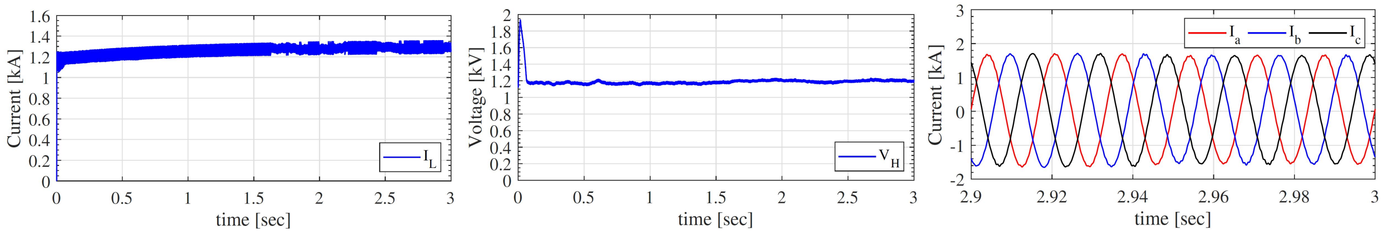
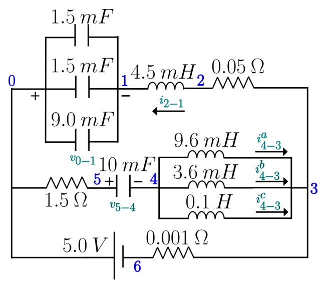


Fig. 2. PV inverter module architecture.



(a) (b) (c)

Fig. 3. PV inverter module simulation results: (a) dc-dc converter's inductor current (b) dc-ac inverter's dc-link voltage (c) dc-ac inverter's output current.



R	2	3	0.05
C	5	4	10e-3
R	0	5	1.5
L	2	1	0.45e-2
C	0	1	1.5e-3
C	0	1	1.5e-3
L	4	3	9.6e-3
L	4	3	3.6e-3
C	0	1	9.0e-3
L	4	3	0.1
V	0	6	5.0
R	6	3	0.001

(a) (b)

Fig. 4. (a) Circuit diagram and (b) netlist of the circuit under study.

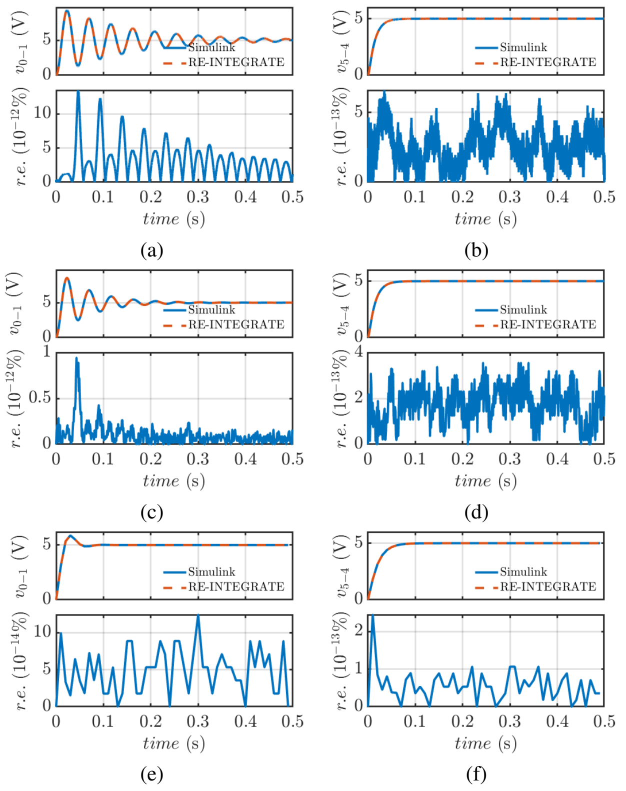


Fig. 5. Results obtained from Simulink & RE-INTEGRATE EMT using stepsize 0.1 ms in (a) and (b), 1 ms in (c) and (d), & 10 ms in (e) and (f).

sufficient damping, leading to the under-damped dynamics of v_{0-1}. Further, as seen from Fig. 5, the oscillations decay faster as the stepsize is increased. This is due to the stiff decay property which enables the implicit Euler method to attain the steady state faster by ignoring the transients, when a higher stepsize is used.

V. Conclusion

In this paper, the DAE solvers and the automation processes currently employed for the newer generation EMT simulation tool called RE-INTEGRATE EMT are documented. A brief overview about the different components of RE-INTEGRATE EMT is provided. Two different use-cases are also reported that were implemented using the DAE solvers outlined in this study. From the simulation results, it is observed that the DAE solvers used are accurate for EMT simulations across different stepsizes as well as different stiffness ratios. Further, the automation processes employed within RE-INTEGRATE EMT are found to significantly reduce the manual intervention that was previously required to discretize the DAEs defining the power system components in EMT simulation models, lending itself to simulation of large-scale power networks.

Acknowledgment

The authors would like to thank Alireza Ghassemian from Department of Energy (DOE) Advanced Grid Modeling (AGM) program for his guidance and support in this work.

References

[1] S. Debnath, P. Marthi, J. Choi, and *et al.*, "EMT simulation of large PV plant & power grid for disturbance analysis," in *2023 IEEE PES Innovative Smart Grid Technologies Latin America (ISGT-LA)*, 2023.

[2] H. W. Dommel, "Digital computer solution of electromagnetic transients in single-and multiphase networks," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-88, no. 4, pp. 388–399, 1969.

[3] [Online]. Available: https://www.headmyshoulder.github.io/odeint-v2/

[4] A. C. Hindmarsh, P. N. Brown, K. E. Grant, , and *et al.*, "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 363–396, 2005.

[5] D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos, "Enabling new flexibility in the sundials suite of nonlinear and differential/algebraic equation solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 48, no. 3, pp. 1–24, 2022.

[6] Q. Xia, S. Debnath, and J. Choi, "Assessing the feasibility of bordered block diagonal reordering in power system matrices using fully convolutional network," in *2024 IEEE Power Energy Society General Meeting (PESGM)*, 2024.

[7] J. Choi and S. Debnath, "Linear solvers for collector systems of generalized large-scale inverter-based resources," in *2024 IEEE Power Energy Society General Meeting (PESGM)*, 2024.

[8] Q. Xia, K. Kurte, and S. Debnath, "Reinforcement learning-based approach for emt automation of large-scale pv plants," in *2024 IEEE Power Energy Society General Meeting (PESGM)*, 2024.

[9] D. R. Reynolds, D. J. Gardner, C. S. Woodward, and R. Chinomona, "ARKODE: A flexible IVP solver infrastructure for one-step methods," *arXiv preprint arXiv:2205.14077*, 2022.

[10] S. Debnath and M. Chinthavali, "Numerical-stiffness-based simulation of mixed transmission systems," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 12, pp. 9215–9224, 2018.

[11] J. Choi and S. Debnath, "Electromagnetic transient (emt) simulation algorithm for evaluation of photovoltaic (pv) generation systems," in *2021 IEEE Kansas Power and Energy Conference (KPEC)*, 2021.

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

DPsim—A dynamic phasor real-time simulator for power systems

Markus Mirz [*], Steffen Vogel, Georg Reinke, Antonello Monti

Institute for Automation of Complex Power Systems, RWTH Aachen University, Aachen, Germany

ARTICLE INFO

Article history:
Received 17 December 2018
Received in revised form 12 April 2019
Accepted 21 May 2019

Keywords:
Real-time simulation
Dynamic phasors
Co-simulation

ABSTRACT

DPsim is a real-time capable solver for power systems that operates in the dynamic phasor and electromagnetic transient (EMT) domain. This solver primarily targets co-simulation applications and large-scale scenarios since dynamic phasors do not require sampling rates as high as EMT simulations do. Due to the frequency shift introduced by the dynamic phasor approach, the sampling rate and rate of data exchange between simulators can be reduced. DPsim supports the Common Information Model (CIM) format for the description of electrical network topology, component parameters and power flow data and it is closely integrated with the VILLASframework to support a wide range of interfaces for co-simulation. Simulation examples demonstrate the accuracy of DPsim and its real-time performance for increasing system sizes.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_244
Legal Code License	GPLv3
Code versioning system used	git
Software code languages, tools, and services used	C++, python
Compilation requirements, operating environments & dependencies	DPsim can be compiled for Linux, OSX and Windows. The main dependencies are: gcc, Eigen, Python, CIM++, VILLASnode, Sundials. A Dockerfile with all dependencies is included in the repository.
If available Link to developer documentation/manual	https://fein-aachen.org/projects/dpsim/
Support email for questions	mmirz@eonerc.rwth-aachen.de

Software metadata

Current software version	v1.0.0
Permanent link to executables of this version	https://hub.docker.com/r/rwthacs/dpsim
Legal Software License	GPLv3
Computing platforms/Operating Systems	Linux docker container
Installation requirements & dependencies	Only a docker installation is required.
If available, link to user manual	https://fein-aachen.org/projects/dpsim/
Support email for questions	mmirz@eonerc.rwth-aachen.de

1. Motivation and significance

DPsim introduces the dynamic phasor (DP) approach to real-time power system simulation. The motivation is to remove the requirement of proportionality between the simulation time step

[*] Corresponding author.
E-mail address: mmirz@eonerc.rwth-aachen.de (M. Mirz).

and the highest frequency of simulated signals. Especially for power electronics and geographically distributed real-time simulation, this is an interesting feature. Currently, the focus of DPsim is more on the application in distributed real-time co-simulation than power electronics. The idea is that the larger the simulation step, the smaller the impact of the communication delay between simulators, which are geographically distributed.

Distributed real-time co-simulation is motivated by large system simulations requiring more simulation capacity than is locally available and by the possibility of Hardware-In-the-Loop (HIL) testing with hardware under test and simulators at different locations. Using dynamic phasors for this application is a fairly recent development although the dynamic phasor, or the envelope function concept, is well known and was introduced in power electronics analysis in [1] as a generalized state space averaging method.

The authors of [2] have developed a distributed real-time simulation laboratory by applying a communication platform as a simulator-to-simulator interface in order to enable remote and online monitoring of interconnected transmission and distribution systems. Each simulator carries out simulations in time domain, while the variables exchanged at the interconnected nodes, i.e. decoupling point, are in the form of time-varying Fourier coefficients. The electromagnetic transient (EMT) values cannot be exchanged at every simulation step due to the communication delay. This delay may be directly incorporated into the physical power system model using traveling wave transmission line models [3]. However, electromagnetic waves travel about 15 km in 50 μs, a time which equals the typical step time in real-time EMT power system simulation. This means that a communication delay in the range of milliseconds would have to be compensated with a line length of several hundred kilometers which may require large and unrealistic changes to the original system model. Therefore, this method is suited for applications on local simulation clusters where the delay is only few simulation steps, but not for internet-distributed simulation, where the expected delay is often tens of milliseconds. In the latter case, the insertion of a transmission line model with the required parameters into the model would have a severe impact on the behavior of the system. Without compensation, the communication delay might cause large errors and even instability of the simulation as shown in [4] for a delay of more than 30 ms.

Starting from the concept presented in [2], we propose interfaces and system-wide simulation state variables based on dynamic phasors. This approach has two benefits explained in [5], which presents a comparison of DP and EMT simulations conducted in DPsim. First, it takes advantage of the computational efficiency of dynamic phasors in scenarios that involve bandpass signals with large center frequencies, as in the case of switching power electronics. Secondly, it increases the simulation time step thus reducing the difference between the local simulation time step size and the round trip time between simulators.

The approach employed in [2] requires the extraction of phasor information from the EMT signals. Therefore, the transparency of the interface is not guaranteed since the interface algorithm may alter the exchanged signals. This extraction step is eliminated by using dynamic phasors as state variables. Because of these features, DPsim is now part of the distributed real-time co-simulation platform described in [6], which is also the base for the experiment described in [2]. A first example of distributed real-time co-simulation using DPsim is presented in [7]. This example shows the impact of the latency in geographical distributed simulation on the results and how the latency can be modeled a priori to running actual simulations.

Besides, there are other relevant open-source initiatives in the field of power system simulation to be mentioned. In particular the Modelica community is very active in adapting Modelica environments for large scale power system cases [8,9] and providing comprehensive libraries of models [10,11]. These initiatives aim to enable large scale, fast simulation of power systems using open-source components but the focus is slightly different compared to the work presented here. The primary objective of DPsim is to assure a deterministic time step in terms of simulated and computation time to provide real-time capability required for Hardware-in-the-Loop (HIL) experiments. This is why in DPsim higher order solver methods are avoided and the system is split into subsystems, which are solved separately. The aforementioned related work does not seem to rely on such compromises since a deterministic time step is not required. While Modelica allows a convenient definition of physical models, DPsim does not intend to provide a solution in this regard. The idea is rather to extend the set of available models in DPsim by generating C-code from Modelica models to represent components connected to the network. These can be linked to DPsim and solved by the ODE solver that was integrated into DPsim for this purpose.

Another related work which is not open-source but also DP based is described in [12]. Similarly to DPsim, the authors have developed a simulator based on the DP approach. However, the focus does not seem to be fast simulation or real-time simulation since the simulator was developed in Python and there are no measures described in order to speed up the simulation. Furthermore, the validation is focusing on the correctness of the simulation rather than simulation speed.

2. Software description

The main theoretical building blocks of DPsim are the dynamic phasor concept and the modified nodal analysis (MNA). Dynamic phasors [13] have various names in scientific literature. Depending on the field and application, they are known as generalized averaging method [1], shifted frequency analysis [14], equivalent envelope [15,16] or baseband signal [17]. Here, the term dynamic phasors is selected because it is widely known in the power system community. The basic idea of dynamic phasors is to approximate a time domain signal x with a Fourier series representation as shown in (1)

$$x(\tau) = \sum_k X_k(t) e^{jk\omega_s(\tau)} \tag{1}$$

where $\tau \in (t - T, t]$. The kth coefficient is determined by

$$X_k(t) = \langle x \rangle_k(t) = \frac{1}{T} \int_{t-T}^{t} x(\tau) e^{-jk\omega_s(\tau)} d\tau \tag{2}$$

where ω_s is the fundamental system frequency and $k\omega_s$ are its harmonics.

MNA is used for the representation of the network as a linear equation system, whereas complex components, such as electrical machines, connected to the network are treated by a separate ODE solver. Depending on the simulation scenario, the admittance matrices of the required network topologies can be preprocessed, i.e. factorized, before simulation start to guarantee a deterministic simulation time step.

The simulation kernel of DPsim is extended with interfaces to support different use cases such as circuit or system simulation, batch simulation, co-simulation and HIL testing. DPsim supports the Common Information Model (CIM) [18] as native input for the description of electrical network topologies, component parameters and load flow data, which is used for initialization. Users can interact with the simulation kernel via Python bindings, which can be used to script the execution, schedule events, change parameters and retrieve results. Python scripts have been proven an easy and flexible way to codify the complete workflow of a

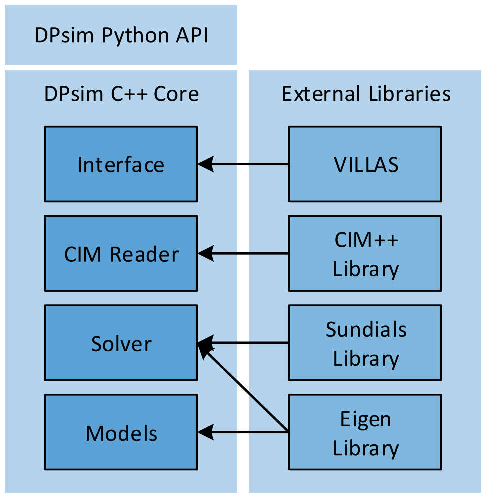


Fig. 1. Overview of DPsim's main components and dependencies on external libraries.

simulation from modeling to analysis and plotting, for example in Jupyter notebooks using Numpy and Matplotlib. Furthermore, DPsim supports co-simulation and interfaces to a variety of communication protocols of commercial hardware via the integration of DPsim with the *VILLASframework* [19], that enables large-scale co-simulations and HIL experiments.

2.1. Software architecture

2.1.1. Module structure

The DPsim simulation kernel and its component models are implemented in the C++ programming language. The availability of good compilers and highly optimized software libraries, such as *Eigen* [20], *Sundials* [21] and *VILLASnode*, which is part of the VILLASframework, for C++ were key factors for this decision.

The core of DPsim consists of models and solvers as depicted in Fig. 1. Interfaces to other software, hardware or data are supported through external libraries. Grid data in CIM standard format is imported using the *CIM++* library [22]. Communication with other software, e.g. real-time simulators, control and monitoring software, as well as hardware is provided by the VILLASframework. For linear algebra operations, DPsim uses the Eigen library. The MNA solver of DPsim uses the Eigen LU factorization and *Eigen::Dense/Sparse::Matrix* are the standard data types for all matrix variables. The Sundials solver package is included in DPsim to provide more complex ODE solvers for components connected to the network.

Compilation from source code requires a Git, a C++11 compiler and CMake 3.6. Tested compilers are Clang, GCC, MSVC and Intel's ICC. As a base operating system Ubuntu 18.04 LTS or Fedora 29 are recommended. For real-time execution a Linux 4.9 kernel with the *PREEMPT_RT* patch-set is recommended.

2.1.2. Class hierarchy

Simulation is the main interface class for users to control the simulation state. The attributes of a *Simulation* instance hold all the information that specifies one simulation scenario in DPsim. The *Simulation* holds references to the solvers, interfaces and the power system model information. This hierarchy is presented in Fig. 2. The complete class hierarchy diagram can be found in the developer's documentation of DPsim [23]. All solvers inherit from *Solver*, e.g. *MNASolver* and *ODESolver*, and all interfaces from *Interface*, e.g. for VILLASnode. All component models, power system, signal etc., derive from the class *IdentifiedObject*. The main

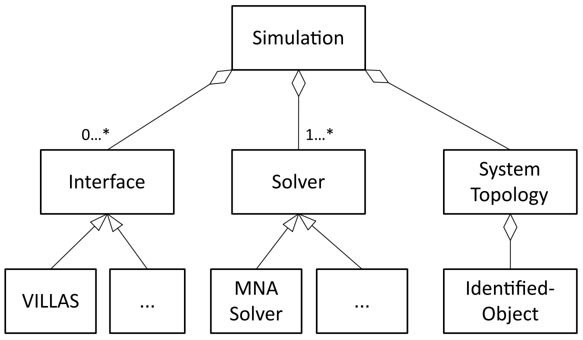


Fig. 2. Diagram of the main classes of DPsim.

attribute of this class is a unique identifier, which is equivalent to the *mRID* in CIM.

2.1.3. Parallelization

In order to utilize the multiple processor cores of modern computer systems and speed up the simulation, the computation of one time step is split up into multiple tasks. These tasks are defined by the various parts of the simulation (like instances of *Solver* and *Interface*). An internal scheduler creates a task graph by analyzing which variables are modified and used by the tasks. This task graph is a directed acyclic graph with nodes representing tasks and edges representing data dependencies. The scheduler uses this graph to distribute the tasks onto multiple threads such that these dependencies are upheld. DPsim implements different scheduling algorithms for this purpose, the simplest of which being level scheduling as used in [24]. This algorithm divides the tasks into ordered levels such that each task only depends on tasks in a previous level. To simulate a time step, these levels are then executed in order by distributing all tasks of one level evenly among the available threads.

To further optimize the simulation performance for large networks when using multiple processors, a decoupling transmission line model can be introduced. In this model a transmission line is represented using equivalent current sources and resistances at the line ends that are not topologically connected as described in [25, ch. 6.2]. Instead, the ends are indirectly connected by updating the values of the sources based on the values on the other end only after a delay τ, which depends on the line's parameters. As subsystems that are connected using this line model are not connected in a strict topological sense, they can be solved independently and in parallel. This significantly reduces the computational effort to simulate large systems.

2.2. Software functionalities

DPsim supports both dynamic phasor and EMT simulation. Furthermore, DPsim is optimized for real-time simulation but it is also possible to run the same simulation scenario offline meaning that the simulation is executed as fast as possible. These different simulation modes are compatible with the user and simulation interface which are covered in the following.

2.2.1. User interface

Network models can be directly defined in the C++ code. This technique is complemented by a CIM importer, which allows the user to directly load network models from CIM-XML files. This proves to be an adequate form to describe network topology and component parameters, which are required by the solver. This CIM importer relieves the user from defining the model in plain C++ code. However, CIM-XML is not suitable for the definition of more complex simulation scenarios with time varying parameters or topology changes caused by contingencies in the system (e.g. breaker events or faults). Therefore, DPsim features Python bindings to most parts of the C++ programming interface. A scripting language like Python is used to define scenarios by leveraging the flexibility of a general purpose imperative language. This allows the user to write a single Python script to:

- Describe the network topology and parameters
- Load a network from CIM data and optionally extend it
- Define a simulation scenario with events or parameter changes
- Execute the simulation and analyze or plot the simulation results

2.2.2. Simulation interface

Interfacing the simulation kernel is desirable for multiple reasons:

- Real-time exchange of simulation signals for co-simulation or HIL testing
- User interface for online monitoring and control of the simulation
- Logging of simulation results for offline analysis
- Import of time series data, for example load and production profiles

For commercial simulation tools these interfaces are a major selling point as new protocols and standards and DAQ cards are introduced continuously. Their implementation and maintenance is time consuming and seemingly never ending. By design, DPsim tries to avoid this pitfall by leaving the implementation of interfaces, data formats and protocols to a separate project. VILLASnode, a component of the VILLASframework project, handles input/output as well as translation between different protocols. DPsim focuses on solving the system model and provides only a single type of interface, shared memory, to the VILLASnode gateway. Interfaces to external systems, databases, files or the web interface are then handled by the wide range of supported interfaces of the VILLASnode gateway, which in this case acts as a proxy between the shared-memory interface to DPsim and the outside world. Responsibilities are clearly separated. This allows the development of new interfaces without having to modify the simulation kernel itself.

In addition, DPsim can use the VILLASnode interfaces for co-simulation with other simulators or remote DPsim instances. In such a scenario, DPsim is usually coupled using an Ideal Transformer Model (ITM). Fig. 3 shows a decoupled model, which exchanges voltages in one and current signals in the opposite direction to control respective sources. For a phasor simulation, the exchanged signals are complex-valued attributes which are passed via the shared-memory interface to VILLASnode which further sends them to a remote simulator using one of VILLAS' supported protocols (e.g. MQTT, UDP, ZeroMQ, …). For geographically distributed simulations, VILLASnode can implement interface algorithms to compensate for the inherent communication latencies when executed in real-time over a high latency connection such as the internet. Alternatively, the implementation of a Discrete Fourier Transform (DFT) in VILLASnode allows for a coupling of the phasor-based DPsim with other EMT-based simulation tools like OPAL-RT or RTDS.

DPsim exchanges simulation data with the VILLASnode gateway via a shared-memory region. The execution of DPsim and VILLASnode as independent processes is crucial in real-time simulation scenarios as the main simulation kernel must not be interrupted by background activities such as data logging to a possibly blocking database.

Furthermore, DPsim has its own simple logging module to write results to CSV files for archival and post processing. This method is easy to setup and convenient for small simulations where post processing and analysis of simulation results is done in MATLAB or Python. In the long term, the internal CSV logging functionality is planned to be incorporated into VILLASnode and enhanced by the support of additional data formats like HDF5 as used by MATLAB.

3. Implementation and empirical results

To complement the previous overview of DPsim's architecture, the next subsection explains implementation specifics affecting the real-time performance of DPsim. The real-time performance of DPsim is demonstrated in the following subsection. The remaining two subsections demonstrate the correctness of the solution computed by DPsim against Matlab Simulink. The first simulation validates only the network solution, which is computed by the MNA solver. The second simulation features a combination of the MNA solver for the network solution and an ODE solver for the numerical integration of the synchronous generator equations.

3.1. Implementation details

Only a compiled language like C++ with minimal runtime overhead is suitable for the implementation since DPsim is targeting simulation time steps in the range of milliseconds to microseconds on off the shelf computing hardware. Great care was taken to avoid memory allocation during the actual simulation. Whenever possible, DPsim utilizes low order integration methods and avoids iterative solver strategies to minimize computation time. That is why the network part is handled by the MNA solver specifically developed for DPsim.

DPsim is compatible with Windows, macOS and Linux operating systems. Eventually, the operating system configuration can have a large impact on the real-time performance. To guarantee real-time execution, DPsim leverages several Linux real-time features such as the real-time capable *SCHED_FIFO* scheduler, real-time signals, the *timerfd* interface, or control groups (*cgroups*). Many of these features are nowadays incorporated in the standard Linux kernel but have their origin in the *PREEMPT_RT* patch-set. The patch-set is slowly integrated into the mainline kernel but still exists and can be applied to further improve the real-time performance by enabling preemption of critical sections such as interrupt handlers. The capability to preempt critical sections in the operating system kernel reduces the overall system latency and therefore helps to ensure strict deadlines at each time-step interval.

Real-time execution on Windows or macOS is not supported. Best real-time performance was achieved on a recent Intel x86_64 multi-core machine with optimized BIOS settings to avoid interruptions of the system by the System Management Mode (SMM). To do so, DPsim execution threads are isolated from remaining system processes using Linux's *cgroup* feature. This reduces the impact of background jobs in the system on the real-time performance.

As a good start for real-time optimizations, we recommend Redhat's Real-time Guide [26] with its *tuned* tool and the *realtime* profile. An updated list of optimization options can be found in the DPsim documentation [23].

To control the time step, DPsim is using *timerfd* interface in Linux environments. The *timerfd* interface allows for the configuration repetitive interval and one-off timers. It uses blocking file descriptors to suspend the execution of the simulation loop until the beginning of the next interval. This approach is more

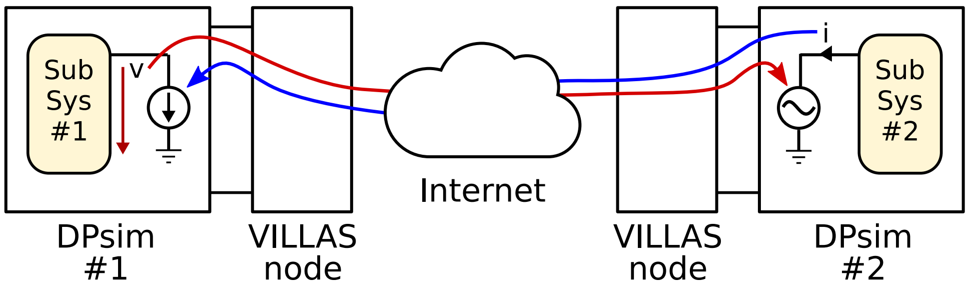


Fig. 3. VILLASnode as a gateway for distributed co-simulation with DPsim.

efficient than the use of the more common *timer* interface, which relies on signals. At the same time, *timerfd* is more accurate as calls to *sleep* or *nanosleep* as they suffer of a lingering drift to non-equidistant execution intervals. The *timerfd* is also used to schedule the synchronized start of distributed simulations as described in the introduction.

Shared-memory is a common method for inter-process communication (IPC) used on symmetric multi-processing (SMP) machines. It enables user processes to exchange data without the involvement of the OS kernel. Shared-memory IPC allows both processes, the solver and the gateway, to be executed in parallel while streaming their data with minimal latency over a queue in the shared memory region. The queue is implemented as a lock free multiple producer/multiple consumer (MPMC) queue and relies on atomic operations of the processor to facilitate synchronization of the DPsim and VILLASnode processes. The lockfree queue is a thread safe data structure. It is used to pass samples of simulation data between the involved processes. As such *message passing* is used as the main paradigm to avoid data races. During the initialization phase, semaphores are used to avoid race conditions in the setup of the shared-memory regions. The absence of the operating system in the communication is crucial to avoid costly context switches, which have to be avoided in a real-time context. Using the shared-memory interface, the DPsim simulation loop can run uninterrupted in a high priority process. At the same time, VILLASnode gets assigned a lower priority for handling of possibly blocking disk or database accesses. In a hardware-in-the-loop simulation it might be necessary to have hard real-time capable interfaces to the real world. For this purpose, DPsim supports an arbitrary number of shared-memory interfaces at the same time. This allows the user to configure one VILLASnode process with a high priority for the control of PCIe FPGA or DAQ cards, and at the same time another VILLASnode process for low priority logging of simulation data in the background.

3.2. Real-time performance evaluation

DPsim is specifically designed for real-time simulation. To assess the effect of system size on the real-time performance of DPsim, a simple test network was copied multiple times and connected with additional transmission lines. Fig. 4a shows the WSCC 9-bus system, which was used for this purpose. The copies were connected in a ring-like topology using additional transmission lines at the nodes 5, 6, and 8. The average wall clock time needed to simulate one time step was measured for a simulation time period of 0.1 s with a time step of 100 µs. Each measurement was further averaged over ten simulations of the same system. The measurements were performed on a system running Ubuntu 16.04 on an Intel Xeon Silver 4114 processor featuring ten cores clocked at 2.2 GHz. The results are shown in Fig. 4b for two different configurations: For the normal simulation, the additional transmission lines were modeled using the Pi model and only one thread was used. For the parallel simulation, the decoupling transmission line model was used for the additional lines and ten threads were employed. As it can be seen, the use of multiple threads and the special transmission line model greatly reduces the wall clock time necessary for simulating a single time step. Even for 40 copies of the original system (resulting in a system size of 360 nodes), the wall clock time per step stays under the simulation time step of 100 µs, thus allowing for real-time simulation. For a small number of system copies, DPsim supports simulation time steps of about 10 µs, which is a typical time step for commercial EMT simulators. However, it should be noted that such small time steps are not the aim of DPsim since the simulation is conducted in DP and not EMT.

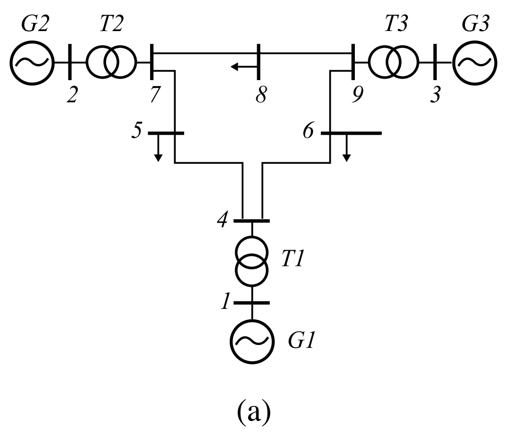
3.3. Validation of the MNA network solver

The next simulation case demonstrates the accuracy of the MNA network solver compared to Simulink results. Both simulators DPsim and Simulink are run with a simulation time step of 100 µs. It can be seen that for such small time steps DP simulations yield the same results as EMT. The larger the time step, the more will the results degrade. It is shown in [5] that the degradation of the results with larger time steps is smaller when using the DP approach compared to EMT.

Fig. 5a shows the simulated circuit, which is composed of one current source of 10 A, two resistors of 1 Ω, an inductor of 1 mH and a capacitor of 1 mF. The voltage source is set to its nonzero peak value at the beginning because it is following a cosine with zero phase shift. Therefore, the system is not starting from steady-state and a transient can be observed. The DPsim dynamic phasor results are transformed to time domain values and compared against Simulink EMT results. As can be seen from Fig. 5b, the results match.

3.4. Validation of the ODE solver for components

The following example compares the results of Simulink and DPsim for a three phase synchronous generator fault. Here, the simulation time step is 50 µs for DPsim and Simulink. As in the previous simulation case, the DP approach would allow for larger time steps than EMT. A comprehensive study investigating this property and featuring synchronous generator models is presented in [27]. Initially, the load is 300 MW and the fault is applied at 0.1 s. The generator parameters are taken from example 3.1 in [28]. As in the previous example, the dynamic phasor results are transformed to time domain values before the comparison. Again, it is visible that the DPsim results match the Simulink results except when the fault is cleared (see Fig. 6). In contrast to DPsim, the fault in Simulink is not immediately cleared but at the next zero-crossing.



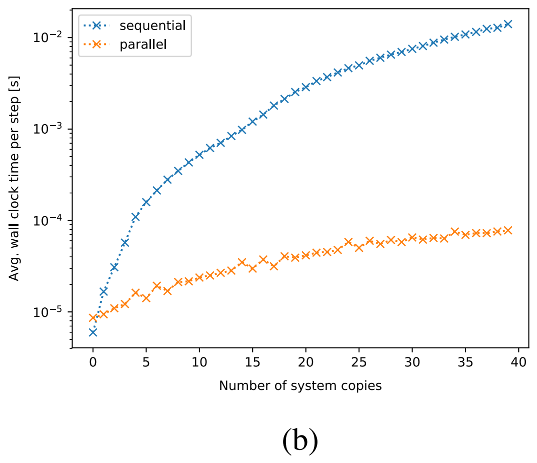
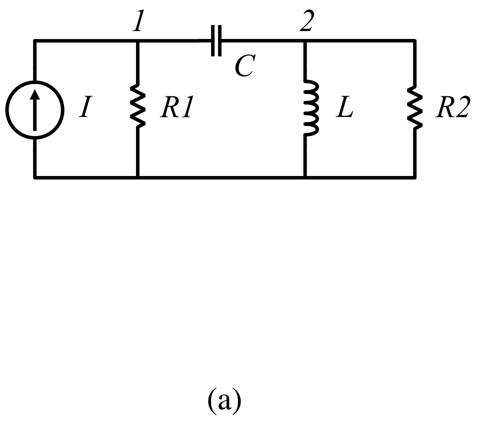


Fig. 4. WSCC 9-bus system (a) and average wall clock time per step (b).



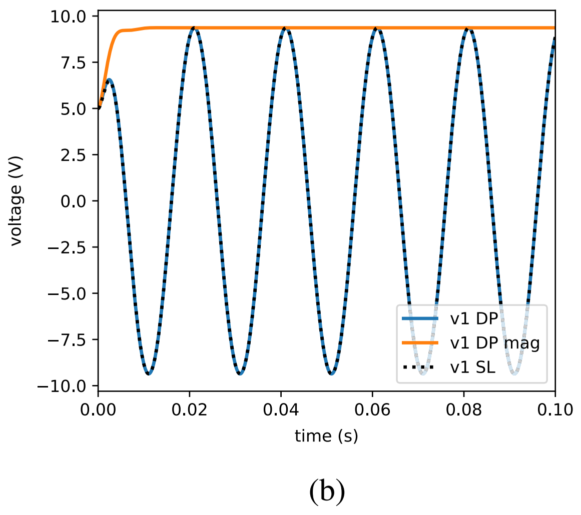


Fig. 5. Example circuit (a) and DPsim dynamic phasor and Simulink EMT simulation results for node 1 (b).

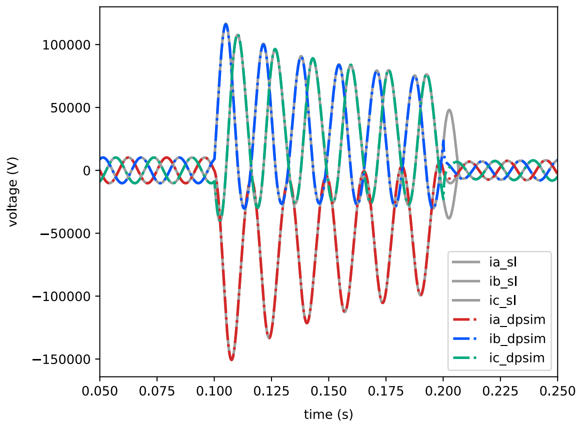


Fig. 6. DPsim dynamic phasor and Simulink EMT results for the synchronous generator three-phase fault example.

4. Illustrative examples

As mentioned in Section 2.2, there are two ways to define a circuit topology for DPsim: coding the topology using Python or C++ or importing it from CIM. The two options are presented by means of two examples: a circuit and a small power system. The first example presented in this section demonstrates the definition utilizing Python while the second example takes advantage of the CIM import function.

4.1. Defining a circuit simulation in python

The circuit of the previous section, Fig. 5, is taken as an example to demonstrate how circuits can be defined using DPsim's Python interface. The topology can be created in Python as depicted by Listing 1.

Listing 1: Python code to define a circuit.

```
# Nodes
gnd = dpsim.dp.Node.GND()
n1 = dpsim.dp.Node('n1')
n2 = dpsim.dp.Node('n2')

# Components
cs = dpsim.dp.ph1.CurrentSource('cs')
cs.I_ref = complex(10,0)
r1 = dpsim.dp.ph1.Resistor('r_1')
r1.R = 1
c1 = dpsim.dp.ph1.Capacitor('c_1')
c1.C = 0.001
l1 = dpsim.dp.ph1.Inductor('l_1')
l1.L = 0.001
r2 = dpsim.dp.ph1.Resistor('r_2')
r2.R = 1
```

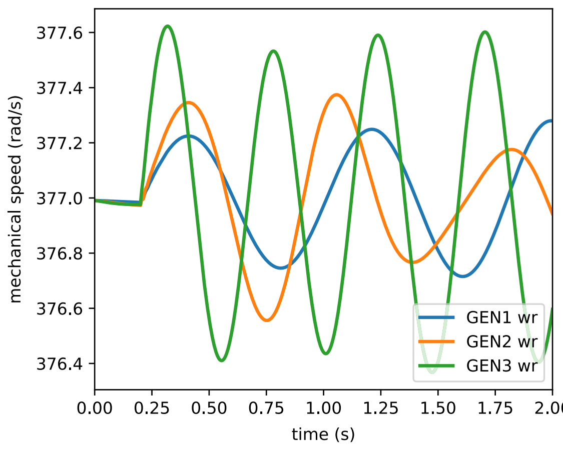


Fig. 7. WSCC 9-bus system mechanical speed simulation results after fault.

```
# Connections
cs.connect([gnd, n1])
r1.connect([n1, gnd])
c1.connect([n1, n2]);
l1.connect([n2, gnd]);
r2.connect([n2, gnd]);

system = dpsim.SystemTopology(50, [gnd, n1, n2], [cs, r1, c1, l1, r2]);
sim = dpsim.Simulation('circuit', system, timestep=0.0001, duration=0.1)
sim.start()
```

First, the nodes and components are declared and parameterized. Then, the connections between components and nodes are set and in the following step all network objects are added to the system topology. Finally, the system topology and parameters such as time step and final time can be used to create a simulation instance, which can be started, stopped and stepped through. Optionally, initial voltages could be assigned to the nodes. Since this is not the case here, the initial voltages are set to zero.

4.2. Simulating a power system defined in CIM

The next example presents the CIM loading functionality, which is used to read the data of the WSCC 9-bus system as displayed in Fig. 4a. The objects defined in the CIM file, e.g. *Terminal*, *TopologicalNode*, *SynchronousMachine*, are mapped to DPsim objects according to the *CIM::Reader* class of DPsim. The system frequency, 60 Hz, is not defined in the CIM file. Therefore, it needs to be specified before loading the CIM file. Furthermore, a fault is applied to node 9 of the imported system. To implement the fault, the system is extended by a switch connected to node 9, which connects the node to ground with a small resistance after 0.2 s. The switching action is created as an object of type *Event* and added to the *Simulation* instance.

Listing 2: Python code to import a topology from CIM.

```
# Read from CIM
files = glob.glob(' ../../ dpsim/Examples/CIM/WSCC09_RX_Dyn/*.xml')
system = dpsim.load_cim('WSCC9bus', files, frequency=60)

# Get existing nodes
gnd = dpsim.dp.Node.GND()
bus9 = system.nodes['BUS6']

# Add switch
sw = dpsim.dp.ph1.Switch('Switch')
sw.R_open = 1e9
sw.R_closed = 0.1
sw.is_closed = False
sw.connect([ bus9, gnd ])
system.add_component(sw)

sim = dpsim.Simulation('WSCC9bus', system,
timestep=0.0001, duration=2, init_steady_state=True)

sim.add_event(0.2, sw, 'is_closed', True)
sim.start()
```

In Fig. 7, it can be seen how the rotational speed of the generators starts to oscillate after the fault. The oscillation frequency depends on the mechanical inertia and as expected the generator with the largest inertia has the lowest oscillation frequency.

5. Impact

Since DPsim is open source, it can serve as a reference implementation for real-time power system simulators and a common basis for users working with different real-time simulation solutions. Having a common basis facilitates discussions on differences in simulation results of different tools. Besides, DPsim allows for real-time simulation on standard computing hardware, making real-time applications available to a wider range of researchers.

Thanks to its design, DPsim facilitates distributed real-time co-simulation, which promotes collaboration and allows the use of the simulation capacity of geographically distributed laboratories to support large scale scenarios [5]. Distributed real-time co-simulation allows also for better data privacy, enabling cooperation because, in a co-simulation, each entity can keep its data confidential and only exchange boundary variables. This could be interesting for confidential grid data but also black box device models where manufacturers cannot share implementation details.

The dynamic phasor approach is used here in a system level simulation in contrast to component level power electronics applications as in [1]. With an increasing share of power electronics in power systems, this approach supports the investigation of future grids.

DPsim is already used in the EU H2020 research project RESERVE [29] and it was developed as a solution to decrease the difference between communication delay and simulation time step in previous co-simulation projects, e.g. RT-Superlab [2]. DPsim is promoted by the FEIN association that also hosts its code and documentation [23].

6. Conclusions

The presented software project, DPsim, exploits the dynamic phasor approach to overcome the requirement for EMT simulation that the simulation time step needs to be proportional to the highest signal frequency. In doing so, DPsim facilitates distributed real-time simulation and lets users exploit simulation resources in different geographical locations. For this purpose, DPsim is programmed in the C++ language and has its own MNA based network solver. Despite having a C++ core, the DPsim allows for scripting simulations via the python interface and reading grid topologies in the standard CIM format via the CIM++ library. These features are demonstrated in two simulation examples, a circuit and a grid simulation. Likewise, the computational correctness of DPsim and its real-time performance are demonstrated by means of simulation examples. Furthermore, DPsim is tightly integrated with the VILLASframework that offers many interfaces to commercial real-time simulators and hardware.

Declaration of competing interest

We confirm that there are no known conflicts of interest associated with this publication.

Acknowledgments

This work was partly funded by the European Unions Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 727481.

References

[1] Sanders SR, Noworolski JM, Liu XZ, Verghese GC. Generalized averaging method for power conversion circuits. IEEE Trans Power Electron 1991;6(2):251–9.

[2] Monti A, Stevic M, Vogel S, De Doncker RW, Bompard E, Estebsari A, Profumo F, Hovsapian R, Mohanpurkar M, Flicker JD, et al. A global real-time superlab: enabling high penetration of power electronics in the electric grid. IEEE Power Electr Mag. 2018;5(3):35–44.

[3] Schutt-Ainé JE. Latency insertion method (LIM) for the fast transient simulation of large networks. IEEE Trans Circuits Syst I 2001;48(1):81–9.

[4] Stevic M, Monti A, Benigni A. Development of a simulator-to-simulator interface for geographically distributed simulation of power systems in real time. In: Industrial electronics society, IECON 2015-41st annual conference of the IEEE. IEEE; 2015, p. 005020–5.

[5] Mirz M, Estebsari A, Arrigo F, Bompard E, Monti A. Dynamic phasors to enable distributed real-time simulation. In: Clean electrical power (ICCEP), 2017 6th international conference on. IEEE; 2017, p. 139–44.

[6] Vogel S, Mirz M, Razik L, Monti A. An open solution for next-generation real-time power system simulation. In: Energy internet and energy system integration (EI2), 2017 IEEE conference on. IEEE; 2017, p. 1–6.

[7] Mirz M, Vogel S, Monti A. First interconnection test of the nodes in pan-european simulation platform. RESERVE Library; 2017.

[8] Braun W, Casella F, Bachmann B, et al. Solving large-scale modelica models: new approaches and experimental results using openmodelica. In: 12 international modelica conference. Linkoping University Electronic Press; 2017, p. 557–63.

[9] Guironnet A, Saugier M, Petitrenaud S, Xavier F, Panciatici P. Towards an open-source solution using modelica for time-domain simulation of power systems. In: 2018 IEEE PES innovative smart grid technologies conference Europe. IEEE; 2018, p. 1–6.

[10] Casella F, Leva A, Bartolini A. Simulation of large grids in openmodelica: reflections and perspectives. In: Proceedings of the 12th international modelica conference, vol. 132. Linköping University Electronic Press; 2017, p. 227–33.

[11] Baudette M, Castro M, Rabuzin T, Lavenius J, Bogodorova T, Vanfretti L. OpenIPSL: Open-Instance power system library—Update 1.5 to "iTesla Power Systems Library (iPSL): A modelica library for phasor time-domain simulations". SoftwareX 2018;7:34–6.

[12] Martí AT, Jatskevich J. Transient stability analysis using shifted frequency analysis (SFA). In: 2018 power systems computation conference. IEEE; 2018, p. 1–7.

[13] Demiray T, Andersson G, Busarello L. Evaluation study for the simulation of power system transients using dynamic phasor models. In: Transmission and distribution conference and exposition: Latin America, 2008 IEEE/PES. IEEE; 2008, p. 1–6.

[14] Martí JR, Dommel HW, Bonatto BD, Barrete AF. Shifted frequency analysis (SFA) concepts for EMTP modelling and simulation of power system dynamics. In: Power systems computation conference. IEEE; 2014, p. 1–8.

[15] Strunz K, Shintaku R, Gao F. Frequency-adaptive network modeling for integrative simulation of natural and envelope waveforms in power systems and circuits. IEEE Trans Circuits Syst I Regul Pap 2006;53(12):2788–803.

[16] Suárez A. Analysis and design of autonomous microwave circuits, vol. 190. John Wiley & Sons; 2009.

[17] Proakis JG. Digital communications. New York: McGraw-Hill; 1995.

[18] Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base. International Electrotechnical Commission; 2016.

[19] FEIN Aachen e. V., VILLAS framework. http://www.fein-aachen.org/projects/villas-framework. [Accessed 23 November 2018].

[20] Guennebaud G, Jacob B, et al. Eigen v3. 2010, http://eigen.tuxfamily.org.

[21] Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans Math Softw 2005;31(3):363–96.

[22] Razik L, Mirz M, Knibbe D, Lankes S, Monti A. Automated deserializer generation from CIM ontologies: CIM++ - an easy-to-use and automated adaptable open-source library for object deserialization in C++ from documents based on user-specified UML models following the Common Information Model (CIM) standards for the energy sector. Comput Sci Res Dev 2018;33(1–2):93–103.

[23] FEIN Aachen e. V., DPsim. https://www.fein-aachen.org/projects/dpsim/. [Accessed 23 November 2018].

[24] Walther M, Waurich V, Schubert C, Gubsch I. Equation based parallelization of modelica models. In: Proceedings of the 10th international modelica conference. p. 1213–20.

[25] Watson N, Arrillaga J. Power systems electromagnetic transients simulation. IET; 2003.

[26] Red hat enterprise Linux for Real Time 7: Tuning Guide; 2018. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/pdf/tuning_guide/Red_Hat_Enterprise_Linux_for_Real_Time-7-Tuning_Guide-en-US.pdf.

[27] Zhang P, Marti JR, Dommel HW. Synchronous machine modeling based on shifted frequency analysis. IEEE Trans Power Syst 2007;22(3):1139–47.

[28] Kundur P, Balu NJ, Lauby MG. Power system stability and control, vol. 7. New York: McGraw-hill; 1994.

[29] RESERVE. http://www.re-serve.eu/. [Accessed 23 November 2018].