SAND2017-6056C

Open Source Tools for High Performance Quasi-Static-Time-Series Simulation Using Parallel Processing

Davis Montenegro[1], Roger C. Dugan[1], and Matthew J. Reno[2]

[1] EPRI, Knoxville, TN, 37923, USA
[2] Sandia National Laboratories, Albuquerque, NM, 87123, USA

Abstract — **Quasi-Static-Time-Series (QSTS) simulation is a valuable tool for evaluating the behavior of power systems through time. By performing daily, yearly and other time-based simulations, it is possible to characterize time-varying power conversion devices such as photovoltaic panels, storage, loads, and capacitors, among others within the power system. However, depending on the time-step resolution and simulation duration, the sequential simulation may require a considerable amount of computing time to complete. This paper describes the OpenDSS-PM program which is the new Parallel Machine version of EPRI's open-source distribution system simulator program, OpenDSS, to accelerate QSTS simulations using multi-core computers. OpenDSS-PM is used to implement temporal parallelization and circuit solutions with Diakoptics based on actors as techniques to reduce the time required in QSTS. The results reveal that these techniques enable a significant reduction in time using common computer architectures.**

I. Introduction

Quasi-Static-Time-Series (QSTS) simulation is a valuable tool for evaluating the behavior of power systems through time. By performing daily, yearly and other time-based simulations, it is possible to characterize time-varying power conversion devices such as photovoltaic panels (PV), storage, loads, and capacitors, among others within the power system. Particularly in the case of PV systems, their proliferation as alternative power source has generated the need of carrying studies such as PV hosting capacity [1, 2], interconnection studies [3-5] and microgrids among others, which are QSTS based.

However, depending on the time-step resolution and simulation duration, the sequential simulation may require a considerable amount of computing time to complete [6]. This paper presents the use of OpenDSS-PM (Parallel Machine), which is derived from EPRI's open-source Distribution System Simulator, OpenDSS [7], to accelerate QSTS simulations using multi-core computers. OpenDSS-PM is used to implement temporal parallelization and Diakoptics based on actors [8, 9] as techniques to reduce the time required in QSTS. The results reveal that these techniques enable a significant reduction in time using common computer architectures. Faster QSTS simulations will provide distribution engineers with a more accurate understanding of the impacts of solar variability and high penetrations of PV on the distribution system [10].

This paper is organized as follows:

- An introduction to OpenDSS-PM
- A brief description of the Parallelization methods
- Simulation results using OpenDSS-PM parallelization
- Conclusions

II. OpenDSS-PM (Parallel Machine)

The OpenDSS program is an open-source electric power Distribution System Simulator (DSS) for supporting distributed resource integration and grid modernization efforts. OpenDSS was originally developed by Electrotek Concepts, Inc. in 1997 under the name of DSS. Since then, it has acquired an important number of capabilities to support the smart grid analysis, including a wide number of device models and simulation modes for this purpose.

This simulation platform was originally built for execution in a single, sequential process. Each procedure/function is called sequentially to perform a QSTS simulation. The performance that can be achieved is based on the structure of the low level routines, the simplicity of the routines, and the efficiency of the compiler.

EPRI made the program open source in 2008 to encourage efforts in grid modernization by providing a tool to the power industry capable of performing advanced studies. The program's name was changed to OpenDSS to emphasize that it was open source. Since then, EPRI has explored several methods to accomplish parallel processing in OpenDSS, including the parallelization of the whole program using a different interface (the Direct DLL API), the modification of the solver using other programming languages, and other methods.

However, these approaches demand additional complications in the user interface, extra effort from the user, and they will be always tied to a particular interface. As a consequence, the desired features that users are accustomed to, such as the COM interface, would be at risk of being deprecated for this type of processing.

Modern computing architectures are characterized for introducing the concept of multi-core computing [11]. This feature allows the performance of applications to be improved by distributing tasks on multiple cores to work concurrently. This feature in modern computers created the obvious need for taking OpenDSS into a parallel computing simulation suite.

EPRI has evolved OpenDSS into a more modular, flexible and scalable parallel processing platform we are calling OpenDSS-PM based with the following guidelines:

- The parallel processing machine will be interface-independent
- Each component of the parallel machine should be able to work independently

- The simulation environment should deliver information consistently
- The data exchange between the components of the parallel machine should respect the interface rules and procedures
- The user interface for the parallel machine should be easy and support the already acquired knowledge of OpenDSS users

To create the parallel machine, OpenDSS-PM uses the actor model [12-15]. There are several actor frameworks for the Delphi language proposed by various authors; however, the chosen framework had already been developed by the authors to evaluate Delphi's tools for parallelization. Each actor is created by OpenDSS-PM, runs on a separate processor (if possible) using separate threads, and has its own assigned core and priority (real-time priority for the process and time critical for the thread).

The interface for sending and receiving messages from other actors will be the one selected by the user: either the COM interface, the Direct DLL API, or a text script using the stand-alone EXE version of the program. From this interface, the user will be able to create a new actor (instance), send/receive messages from these actors, and define the execution properties for each actor. The properties include the execution core, simulation mode, and circuit to be solved, among others. The actor parallelization concept is presented in Fig. 1.

III. The Parallelization Techniques

The parallelization techniques utilize multiple independent computing resources to decompose the simulation complexity. This decomposition can be done by distributing the work in time or by simplifying the power flow problem complexity using simpler representations of it.

As a result, the amount of time required for solving a large number of simulation steps is expected to be reduced. In this paper two methods are explored: *Temporal parallelization* and *Diakoptics* based on actors. The first one seeks to distribute temporal segments of the total simulation task into multiple actors to complete the whole simulation.

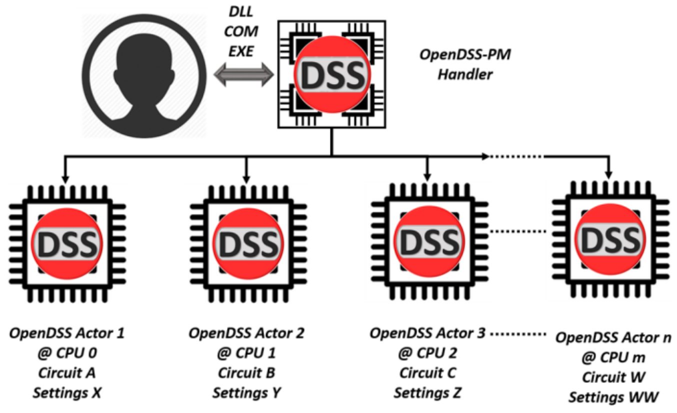
DLL
COM
EXE
DSS
OpenDSS-PM Handler
OpenDSS Actor 1 @ CPU 0 Circuit A Settings X
OpenDSS Actor 2 @ CPU 1 Circuit B Settings Y
OpenDSS Actor 3 @ CPU 2 Circuit C Settings Z
OpenDSS Actor n @ CPU m Circuit W Settings WW

Fig. 1. OpenDSS-PM general description.

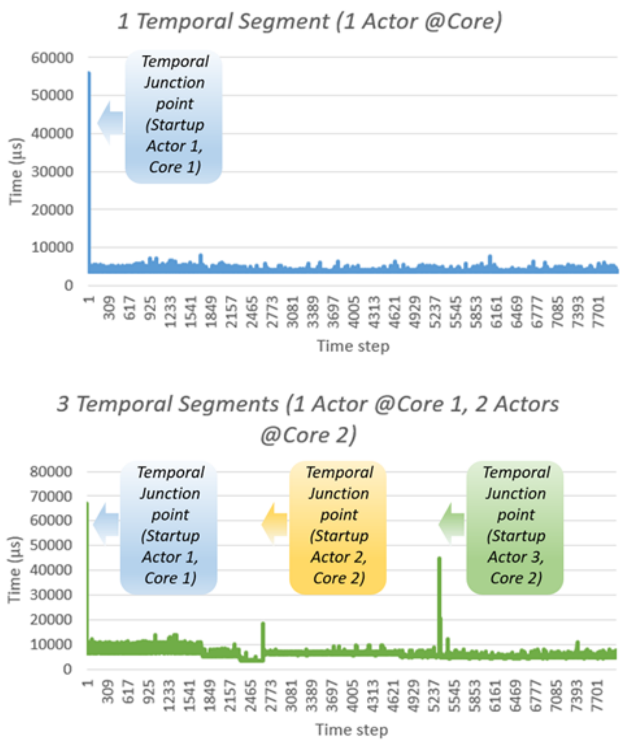

Fig. 2. Temporal Parallelization concept. The top figure is the standard QSTS, and the bottom figure below demonstrates temporal parallelization of QSTS.

The second one decomposes the interconnected circuit into smaller and simpler sub-circuits, each of these systems will be assigned to an actor to find a partial solution that will be complemented in a later stage.

A. *Temporal parallelization*

Temporal parallelization is a technique that consists of splitting the simulation into multiple time periods with each period being simulated sequentially and concurrently to each other. Each of the multiple temporal segments will deliver partial results that after being concatenated will deliver a very close result as if the simulation was performed sequentially. This technique is illustrated in Fig. 2.

In this simulation technique the circuit's complexity remains intact, which means that each temporal segment requires a startup simulation time. The startup time is the longest time in a QSTS simulation and will force a non-linear acceleration when using this technique. When applying temporal parallelization, the startup simulation time can be located at the junction points between actors.

Additionally, since distribution power flows are dependent on the previous time-step and there is no information about the state of the control devices at these points before starting the simulation, a systematic error can be introduced. Nevertheless, if the adequate control actions are performed during the startup step the systematic error can be reduced. OpenDSS-PM is designed to perform all the needed control actions at the simulation start up for each concurrent actor to minimize the error and maintain the simulation fidelity.

B. Diakoptics Based on Actors

Diakoptics Based on Actors (*A-Diakoptics*) is the result of combining two computing techniques from different fields. On one hand there is *Diakoptics* [16, 17], which is a mathematical method for tearing networks to create a set of independent sub-networks. Each sub-network can be handled and solved independently and the size of each sub-network is smaller than the interconnected network.

The *Actor model* is an information model for dealing with inconsistency robustness in parallel, concurrent, and asynchronous systems [12, 18]. This information model allows multiple processes to be executed in parallel with information passing through them using messages. As a result, the information consistency can be ensured, avoiding frequent issues when working with parallel processing systems, such as race conditions and memory sub-utilization (when working with multicore processors) among others.

A basic description of A-*Diakoptics* is shown in Fig. 3. The subsystems, and their relationships can be described as a set of matrixes representing the subsystems (Z_{TT}), the link branches between them (Z_{CC}) and their relationship within the network (Z_{CT}). As a result, by using this information and the partial results delivered after solving each subsystem separately and concurrently, it is possible to determine the voltages at the network's nodes by finding a complementary answer as follows:

$$E_T = Z_{TT} I_{0(n)} - Z_{TC} Z_{CC}^{-1} Z_{CT} I_{0(n+m)} \qquad (1)$$

A-Diakoptics is a technique that seeks for simplifying the power flow problem to perform a faster solution at each simulation step; in contrast, temporal parallelization remains the complexity of the problem. Consequently, the total time reduction when performing QSTS will be evident at each simulation step with A-Diakoptics.

IV. Results

For evaluating the performance of OpenDSS-PM in the temporal parallelization, EPRI's Test Circuit 5 [19] is used.

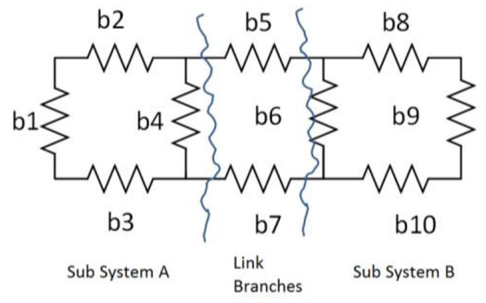

Fig. 3. Physical interpretation of Diakoptics.

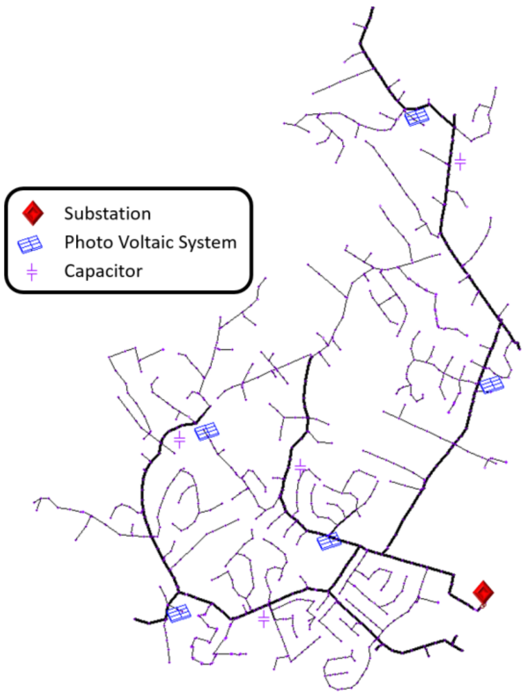

Fig. 4. EPRI Circuit 5.

This circuit is shown in Fig. 4. In this circuit, there are 5 zones where PV systems are concentrated providing a total peak power of 2.5MW to the system (aggregate). The temporal parallelization is performed by considering the number of available CPUs (threads) on the computer where the tests are performed.

A. Temporal parallelization

In this case the power flow is solved 8000 times using the *time* simulation mode and step size of 1h, which is equivalent duration of 11 months and 4 days. These quantities have been arbitrarily selected to present round numbers for this example. The script wrote using the OpenDSS-PM scripting tool is as follows:

```
clearAll
set parallel=No  ! For the compilation stage
compile "C:\ OpenDSS\EPRI_ckt5_3437_nodes\master.dss"
set CPU=0
Solve
set mode=time stepsize=1h number=2000 hour = 0 totaltime=0
NewActor              ! Creates a new actor
compile "C:\ OpenDSS\EPRI_ckt5_3437_nodes\master.dss"
set CPU=2
Solve
set mode=time stepsize=1h number=2000 hour = 2000 totaltime=0
NewActor                              ! Creates a new actor
Compile "C:\OpenDSS\EPRI_ckt5_3437_nodes\master.dss"
set CPU=4
Solve
set mode=time stepsize=1h number=2000 hour = 4000 totaltime=0
NewActor                              ! Creates a new actor
Compile "C:\OpenDSS\EPRI_ckt5_3437_nodes\master.dss"
set CPU=6
Solve
set mode=time stepsize=1h number=2000 hour = 6000 totaltime=0
set parallel=Yes
SolveAll
```

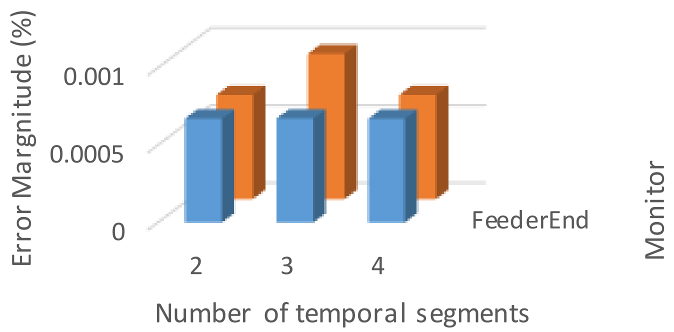

Fig. 5. Error magnitude after separating the simulation in several temporal segments

TABLE I. RESULTS ACHIEVED AFTER SPLITTING THE SYSTEM IN 4 TEMPORAL SEGMENTS

	Actor	CPU	Initial time	Iterations	Time (sec)	Time reduction (%)
OpenDSS			0	8000	28.33	
OpenDSS-PM	1	0	0	2000	9.68	65.83
	2	2	2000	2000	9.43	66.71
	3	4	4000	2000	9.55	66.29
	4	6	6000	2000	9.55	66.29

By using the script presented as example, four actors are created to split the solution time in sets of 2000 time steps (4 temporal segments). This test was performed in a computer with a processor Intel core i7-2720 @ 2.2 GHz. The errors are evaluated for each simulated cases (1, 2, 3 and 4 temporal segments) to quantify the error from temporal parallelization and how its magnitude could affect the results. The results are shown in Fig. 5 and in TABLE I.

The voltage error magnitude is below 1.0e-3 p.u., revealing that the control actions performed at each actor's startup takes the partial simulation variables to an acceptable set of values. The error magnitude becomes non-zero after the first junction point between the different time segments. However, this error is still within acceptable ranges.

Then, using this architecture the system is solved by creating 2, 3 and 4 temporal segments to evaluate the time reduction. This processor has only 2 cores in comparison with the i7 processor used in the previous test. The aim of this test is to highlight the effect of running 2 actors on the same core when trying to perform parallel processing. The results of this test are shown in TABLE II. As can be seen in TABLE II, when setting the affinity of 2 actors to the same processor's core the performance of the actor get decreased.

This an expected result since the core needs to attend 2 processes simultaneously, which is impossible. As a consequence, both actors (threads) get serialized, and the processing time for each one is higher than if the core were dedicated to a single Actor as shown in Fig. 6.

The overhead obtained after splitting the simulation in time can be justified as follows:

TABLE II. RESULTS OBTAINED WITH TEMPORAL PARALLELIZATION USING A SECOND ARCHITECTURE

		CPU	Iterations	Time (sec)	Time reduction (%)	
2 temporal segments						
OpenDSS			8000	31.19		
OpenDSS-PM	Actor 1	0	4000	18.24	35.62	Core 1
	Actor 2	2	4000	17.36	38.72	Core 2
3 temporal segments						
OpenDSS			8000	31.19		
OpenDSS-PM	Actor 1	0	2666	14.12	50.16	Core 1
	Actor 2	2	2666	16.36	42.25	Core 2
	Actor 3	3	2666	16.27	42.57	
4 temporal segments						
OpenDSS			8000	31.19		
OpenDSS- PM	Actor 1	0	2000	14.64	48.32	Core 1
	Actor 2	1	2000	13.76	51.43	
	Actor 3	2	2000	13.73	51.54	Core 2
	Actor 4	3	2000	13.62	51.92	

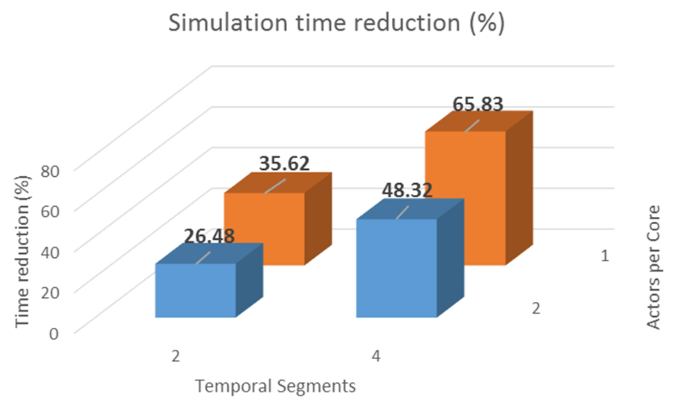

Fig. 6. Effects of having 1 or 2 Actors per Core when performing parallel processing

1. The system's complexity remains intact, this is, the size of the system is the same after and as a consequence, it will require a startup procedure on every tearing/starting point as shown in Figure 5. Depending on the startup point in time, there could be involved more/less control actions on each temporal segment to find an accurate solution.
2. The coordination process taken between OpenDSS-PM and other applications when an actor is launched by OpenDSS-PM, to give to the actor the highest priority can add an additional overhead for the actor startup. This process is performed by the Operating System (OS) and cannot be controlled when using standard OS.
3. Even when the actors are running on separate cores, the processor's power consumption increases due to the dedicated priority set for each actor. Normally the system motherboard takes actions to avoid the processors overheating and to reduce the power consumption.

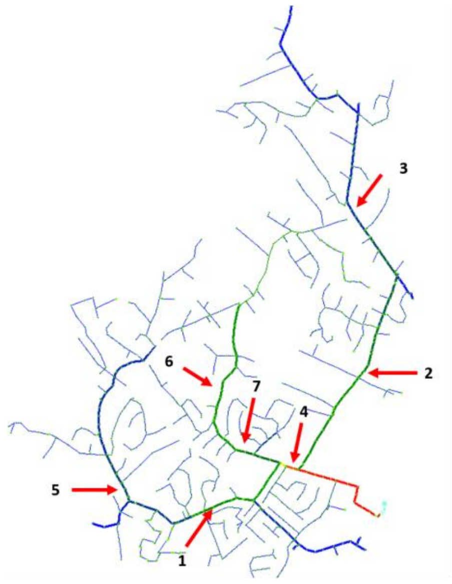

Fig. 7. EPRI circuit 5 and location of the proposed link branches

Depending on the hardware architecture (laptop, desktop PC, Type of Power Supplies Units-PSU), the performance could be affected when running multithread dedicated applications. More PC architectures running with standard operating systems need to be evaluated.

B. *Diakoptics Based on Actors*

Similar parallelization QSTS simulations are performed when using the A-Diakoptics method. From previous works [2], a 70-80% reduction in solution time is expected; however, there will be some variations in time improvements based on the computing environment. In fact, depending on the number of available processor cores available and the number of actors created the results may change.

The same test is performed by using another computer with a processor Intel i5-5200 @ 2.2 GHz CPU. At this point, both computers are laptop computers.

In the case of *A-Diakoptics* the interconnected circuit is teared using 7 link branches as shown in Fig. 7. The link branches were selected to generate multiple configurations and several balanced/unbalanced distribution of nodes. The balance is measured in terms of the amount of nodes handled by each actor. If 2 actors are handling the same amount of nodes, the parallel system is balanced; otherwise, the system is unbalanced in a certain degree.

The first test consists of performing the simulation using multiple combinations of link branches to generate a multiple number of actors. The results are shown in TABLE III. For this case the simulations are performed using a co-simulation environment between NI LabVIEW and OpenDSS-PM.

Depending on the balance, the system performance will be seriously affected. For the case presented, the system is slightly balanced, however, if the unbalance gets higher the solution times can be considerably affected.

Another interesting finding is the fact that the overhead added by the co-simulation platform is significant compared with the performance of the OpenDSS-PM application working in stand-alone mode. To verify this hypothesis the time per step of each actor is totalized after each simulation considering the number of actors used to solve the system. This information is shown in TABLE IV.

The results presented in TABLE IV shows that the actors are indeed asynchronous systems. The total simulation time will be the time of the slowest actor since at each iteration the actors need to be synchronized to perform the extra calculations.

These results also reveal that while working with more actors, the total overhead added by LabVIEW becomes important. This difference becomes more relevant when each actor is executed on a separate core as shown in Fig. 8.

For the results discussed in this report we have used an Intel® Xeon® CPU E5-2650 v4 @ 2.2GHz, while in [8, 9] an Intel® CoreTM i7-4810MQ @ 3.4 GHz was used. This means that working with OpenDSS-PM to perform piecewise methods results in an accelerated simulation experience, which can be used for multiple applications including Real-Time simulation.

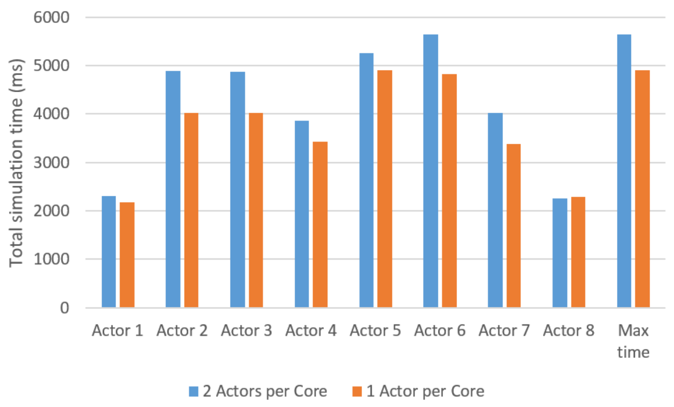

Fig. 8. Total simulation time per actor when allocating 1 or 2 actors per core

TABLE III. RESULT ACHIEVED AFTER TEARING THE SYSTEM IN MULTIPLE ACTORS

Number of actors	Time Elapsed (ms)	Time reduction (%)	Number of Cores used
8	14848	68	4
6	16782	64	3
4	16934	63	2
2	19050	59	1
1	46684	0	1

TABLE IV. SIMULATION TIMES FOR EACH OF THE ACTORS IN OPENDSS-PM

# of Actors	Simulation time per actor (ms) - 8760 time steps							
	A1	A2	A3	A4	A5	A6	A7	A8
8	2307	4896	4868	3870	5255	5647	4023	2264
6	2154	2555	4434	11003	5923	3607		
4	3593	10658	9628	9958				
2	7682	14413						

TABLE V. ERRORS IN PERCENTAGE FOR EACH SIMULATION CASE

Node name	Error (%) per number of actors			
	8 Actors	6 Actors	4 Actors	2 Actors
sourcebus.1	0	0	0	0
sourcebus.2	0	0	0	0
sourcebus.3	0	0	0	0
1023346.1	2.010771	2.012686	2.01203	0.807802
1023346.2	2.0915	2.111095	2.108079	0.966332
1023346.3	2.261792	2.253757	2.236306	1.050958
63657.1	2.642765	2.652466	2.649035	1.039492
63657.2	2.638333	2.667622	2.678754	1.20196
63657.3	2.88666	2.865874	2.835851	1.35616
816.2	2.685346	2.288218	2.120706	1.228647
x_1103251_1.3	3.184148	2.531579	2.637314	1.568667
28243.1	2.778495	2.108859	2.258119	1.084387
39582.1	3.310538	3.294892	2.667483	2.642919
39582.2	3.115652	3.106987	2.556511	2.514462
39582.3	3.658225	3.636021	2.87142	2.83938
x_39760_3.2	2.598807	2.586058	2.175612	1.216944

To evaluate the simulation fidelity and the error introduced by the technique used in this report, several measurements are performed around the circuit. The locations of the measurements are selected randomly. The errors in percentage for each simulation case are shown in TABLE V.

As can be seen in TABLE V, the error gets higher when more actors are used. However, this behavior is due to the lack of synchronism between the control actions and the modifications of the actor's YBUS matrix. Because matrixes Z_{TC}, Z_{CC} and Z_{TC} are calculated at the beginning of the algorithm, if the YBUS matrices of the actors change during the simulation, the error will increase. This source of error suggests that is necessary to investigate methods for applying the non-invasive control actions, avoiding the changing of the actor's YBUS matrix, and thus reducing the systematic error added.

V. CONCLUSIONS

This paper has presented the use of EPRI's OpenDSS-PM (Parallel Machine), to accelerate QSTS simulations using multi-core computers. Temporal parallelization and A-Diakoptics provide new techniques to reduce the time of QSTS simulations without losing simulation fidelity. The paper has discussed in depth the application of these techniques using the standard distribution test feeders that include PV generation with realistic irradiance profiles. With increasing penetrations of distributed energy resources (DER) and distributed PV, the ability to quickly perform QSTS simulations is a crucial aspect of distribution system planning and operations.

OpenDSS-PM is an open source simulation tool available at the OpenDSS website at sourceforge.net. The example presented in this paper are available for NI LabVIEW when installing the OpenDSS-PM library for NI LabVIEW using the VI Package Manager tool [20].

REFERENCES

[1] S. Jothibasu and S. Santoso, "Sensitivity analysis of photovoltaic hosting capacity of distribution circuits," in *2016 IEEE Power and Energy Society General Meeting (PESGM)*, 2016, pp. 1-5.

[2] M. Rylander, J. Smith, D. Lewis, and S. Steffel, "Voltage impacts from distributed photovoltaics on two distribution feeders," in *2013 IEEE Power & Energy Society General Meeting*, 2013, pp. 1-5.

[3] M. Rylander, J. Smith, W. Sunderman, D. Smith, and J. Glass, "Application of new method for distribution-wide assessment of Distributed Energy Resources," in *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*, 2016, pp. 1-5.

[4] M. J. Reno, K. Coogan, R. Broderick, and S. Grijalva, "Reduction of distribution feeders for simplified PV impact studies," in *2013 IEEE 39th Photovoltaic Specialists Conference (PVSC)*, 2013, pp. 2337-2342.

[5] M. J. Reno, M. Lave, J. E. Quiroz, and R. J. Broderick, "PV ramp rate smoothing using energy storage to mitigate increased voltage regulator tapping," in *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*, 2016, pp. 2015-2020.

[6] M. J. Reno, J. Deboever, and B. Mather, "Motivation and Requirements for Quasi-Static Time Series (QSTS) for Distribution System Analysis," presented at the IEEE Power Engineering Society General Meeting, Chicago, 2017.

[7] R. C. Dugan and T. E. McDermott, "An open source platform for collaborating on smart grid research," in *2011 IEEE Power and Energy Society General Meeting, ,* 2011, pp. 1-7.

[8] D. Montenegro, G. A. Ramos, and S. Bacha, "Multilevel A-Diakoptics for the Dynamic Power-Flow Simulation of Hybrid Power Distribution Systems," *IEEE Transactions on Industrial Informatics,* vol. 12, pp. 267-276, 2016.

[9] D. Montenegro, G. A. Ramos, and S. Bacha, "A-Diakoptics for the Multicore Sequential-Time Simulation of Microgrids Within Large Distribution Systems," *IEEE Transactions on Smart Grid,* vol. PP, pp. 1-9, 2016.

[10] J. Deboever, X. Zhang, M. J. Reno, R. J. Broderick, S. Grijalva, and F. Therrien, "Challenges in reducing the computational time of QSTS simulations for distribution system analysis," Sandia National Laboratories SAND2017-5743, 2017.

[11] J. Diaz, C. Munoz-Caro, and A. Nino, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," *IEEE Transactions on Parallel and Distributed Systems,* vol. 23, pp. 1369-1386, 2012.

[12] C. Hewitt, E. Meijer, and C. Szyperski. (2012, 05-15). *The Actor Model (everything you wanted to know, but were afraid to ask)*. Available: http://channel9.msdn.com/Shows/Going+Deep/Hewitt-Meijer-and-Szyperski-The-Actor-Model-everything-you-wanted-to-know-but-were-afraid-to-ask

[13] N. Instruments. (2013, 24, 05). *Actor Framework Template documentation*. Available: http://www.ni.com/white-paper/14115/en

[14] L. Jie, J. Eker, J. W. Janneck, L. Xiaojun, and E. A. Lee, "Actor-oriented control system design: a responsible framework perspective," *IEEE Transactions on Control Systems Technology,* vol. 12, pp. 250-262, 2004.

[15] D. Montenegro, "Actor's based diakoptics for the simulation, monitoring and control of smart grids," Université Grenoble Alpes, 2015.

[16] G. Kron, *Diakoptics: the piecewise solution of large-scale systems*: Macdonald, 1963.

[17] H. H. Happ, *Piecewise Methods and Applications to Power Systems*: Wiley, 1980.

[18] C. Hewitt, "Actor Model of Computation: Scalable Robust Information Systems," in *Inconsistency Robustness 2011, Stanford University*, 2012, p. 32.

[19] J. Fuller, W. Kersting, R. Dugan, and S. C. Jr. (2013, 10-23). *Distribution Test Feeders*. Available: http://ewh.ieee.org/soc/pes/dsacom/testfeeders/

[20] JKI. (2016, 01/06/2017). *VI Package Manager*. Available: https://vipm.jki.net/

This research was supported in part by the DOE SunShot Initiative, under agreement 30691. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525