Scalable Power Grid Transient Analysis via MOR-Assisted Time-Domain Simulations

Jia Wang and Xuanxing Xiong
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago, IL 60616, USA

***Abstract*—Time domain power grid simulations provide accurate estimations of power supply noises for design verifications. Despite extensive researches, it remains a very challenging problem to perform the simulations efficiently – the large power grid sizes demand tremendous computational resources and the causality between consecutive time steps hinders parallel implementations. Frequency domain and model order reduction (MOR) techniques promise scalability, though the solution accuracy may become a concern without trading off running time. In this paper, we present a framework for power grid transient analysis where time domain simulations are assisted by MOR techniques for scalability without losing much of the solution accuracy. Utilizing a direct sparse matrix solver and the multinode moment matching technique, we are able to achieve more than $5X$ speed-up using 16 processor cores distributed over two servers when simulating 1000 time steps for all the six IBM power grid simulation benchmarks, in comparison to the time domain simulations using only the direct solver.**

I. Introduction

Modern design verifications rely heavily on power grid analysis to provide estimations of power supply noises. Time domain simulations based on implicit numerical integration techniques including backward Euler and trapezoidal methods are widely adopted and are usually treated as "golden" for transient analysis. The rapid growth of parallel and distributed computing platforms makes it desirable, though extremely challenging, to build scalable transient analysis tools in order to handle the increasing power delivery complexity.

Time domain simulations depend on a sparse matrix solver to solve the system of linear equations at each time step As this solver contributes to most of the memory usage and the simulation time, it is the major focus for parallellization. When the memory usage of the solver is beyond the capacity of a single machine, hierarchical analysis [21] can be applied to partition the power grid and to distribute the job to multiple machines. For DC analysis where a single time step is of concern, parallel solvers exploiting various feature of power grids and leveraging advanced numerical methods including multigrid [5], [6], fast transform [3], and additive Schwarz method [14], [20] have shown great success over direct solvers based on LU decomposition in both memory usage and running time. However, for transient analysis where the LU decompositions are usually computed at the very beginning, it is very difficult to achieve speed-ups in comparison to the direct solvers when there is enough memory since only a pair of forward and back substitutions are necessary at each time step [12]. Though one may parallelize forward and back substitutions either by partitioning the power grid before LU decompositions [21], [14], [17], or by partitioning the L and U matrices [19], [18], [17], both approaches suffer from their own shortcomings. For the former approach, the number of fill-ins is a concern and if fill-ins are dropped to maintain the sparsity of the L and U matrices, solution accuracy has to be guaranteed via an iterative solver that introduces additional overhead. For the latter approach, the availablility of parallelism is a concern and less than 2X speed-ups were observed [12]. Moreover, it is demonstrated in [17] that even when there is ample parallelism to be exploited, the memory traffic required by multiple forward and back substitutions running in parallel on a single machine could be the limiting factor that prevents better scaling. Overall, although it is possible to parallelize the sparse matrix solver, the causality between consecutive time steps are not removed. The simulation time will remain proportional to the number of the time steps no matter how many machines are available, and the performance on parallel and distributed platforms may be further constrained by the communications required at the boundary of the time steps.

On the other hand, in theory it is possible to obtain power supply noises in a scalable manner – since the power grid is a linear time-invariant (LTI) system, the noise for any node at any time is the convolution of the past inputs and the impulse responses. In practice, frequency domain [22], [8], model order reduction (MOR) [15], [10], [11], and matrix exponential [16] methods have been studied as alternatives to time domain simulations for power grid transient analysis and show promises of scalability as they do not depend on the time steps explicitly. However, these methods are still quite costly if similar accuracy as time domain simulations is desired due to the difficulty in handling current excitations. For frequency domain methods, the overheads in both running time and storage are introduced when converting the independent sources, especially those rapid changing ones, from the time domain into the frequency domain as their spectrums are not bounded. For MOR techniques, the effectiveness degrades when there are many independent sources that cannot be abstracted away, which is usually the case for power grid transient analysis. As independent sources are not a concern for time domain simulations, it is highly desirable if a transient analysis algorithm could combine the advantages of both time domain simulations and LTI techniques for accuracy, efficiency, and scalability.

In this paper, we propose a framework for power grid transient analysis that leverages MOR techniques to assist time domain simulations in order to achieve scalability without affecting accuracy and efficiency. We partition the time steps into intervals and defining for each interval a current excitation that equals to the actual input wthin the interval, but is zero elsewhere. According to the well-known superposition property of LTI systems, the actual noises are the summations of the responses to this set of current excitations. Note that although the responses can be computed independently, to obtain them directly from time domain simulations will not have any benefit – in the worst case the number of the time steps are not reduced. We propose to circumvent this difficulty by applying time domain simulations only to the non-zero parts of the current excitations. MOR techniques are then applied to estimate the responses when the current excitations are zero. From the perspective of MOR techniques, in our framework time domain simulations are applied to circumvent the need to convert independent sources. Combining a direct sparse matrix solver for time domain simulations and the multinode moment matching technique [9] for MOR in our proposed framework, we are able to achieve more than $5X$ speed-up using 16 processor cores

978-1-4799-1071-7/13/$31.00

distributed over two servers when simulating 1000 time steps for all the six IBM power grid simulation benchmarks, in comparison to the time domain simulations using the direct solver along, without applying paralellization of forward and back substitutions. We further note that one of the most important features of our framework is that it is *orthogonal* to most existing parallel and distriubted algorithms and thus it is possible to integrate them into our framework for better scalability.

The rest of this paper is organized as follows. Time domain simulation and multinode moment matching are reviewed in Section II. Our proposed framework is presented in Section III. After experimental results are discussed in Section IV, Section V concludes the paper.

II. Preliminaries

A. Time Domain Simulation

The power grid transient analysis problem seeks a numerical solution of the ordinary differential equation (ODE) in Eq. (1) derived by modified nodal analysis (MNA) [7].

$$\begin{aligned} Gv(t) + Cv'(t) + \mathcal{B}^T i(t) &= I(t), \\ \mathcal{B}v(t) &= \mathcal{L}i'(t). \end{aligned} \tag{1}$$

In the above equation, $v(t)$ and $i(t)$ are unknown nodal noises and branch currents. The independent current sources entering each node are given as $I(t)$. Suppose there are n nodes and l inductive branches in the power grid. Then G and C are $n \times n$ diagonally dominant symmetric matrices obtained from resistor and capacitor stamps, B is a $l \times n$ matrix of $0/1/-1$ representing the directions of the inductive branches and the nodes they incident on, and $\mathcal{L}$ is an $l \times l$ matrix specifying the mutual and self inductances among the inductive branches.

Choosing a time step h, Eq. (1) can be solved by time domain simulations based on implicit numerical integration techniques. Let $v_k = v(kh)$, $i_k = i(kh)$, and $I_k = I(kh)$ for $k = 0, 1, \ldots$. Time domain simulations first solve for v_0 and i_0 using the following DC initial condition,

$$A \begin{pmatrix} v_0 \\ i_0 \end{pmatrix} = \begin{pmatrix} I_0 \\ 0 \end{pmatrix}, \text{ where } A \triangleq \begin{pmatrix} G & \mathcal{B}^T \\ \mathcal{B} & 0 \end{pmatrix}. \tag{2}$$

Successive v_k's and i_k's for $k > 0$ are then solved iteratively, e.g. by the trapezoidal rule as follows,

$$A_h \begin{pmatrix} v_{k+1} \\ i_{k+1} \end{pmatrix} = -B_h \begin{pmatrix} v_k \\ i_k \end{pmatrix} + \begin{pmatrix} I_{k+1} + I_k \\ 0 \end{pmatrix}, \tag{3}$$

where the matrices A_h and B_h are defined as,

$$A_h \triangleq \begin{pmatrix} G + \frac{2}{h}C & \mathcal{B}^T \\ \mathcal{B} & -\frac{2}{h}\mathcal{L} \end{pmatrix}, B_h \triangleq \begin{pmatrix} G - \frac{2}{h}C & \mathcal{B}^T \\ \mathcal{B} & \frac{2}{h}\mathcal{L} \end{pmatrix}. \tag{4}$$

Eq. (2) and (3) are usually solved by direct solvers via a pair of forward and back substitutions assuming the LU decompositions of A and A_h are computed once at the beginning. Note that both A and A_h should not be singular – for power grids, this is guaranteed by a reasonable assumption requiring that there is no inductive loop and that every node has a resistive path to either ground or power supply.

If the memory usage to store the L and U metrices is of concern and there is no mutual inductance, i.e. $\mathcal{L}$ is a diagonal matrix, then Eq. (3) can be transformed into a system of linear equations with the left-hand-side system matrix being diagonally dominant and symmetric [1]. In such case, the memory usage can be reduced by half by Cholesky decomposition, and can be further reduced via preconditioned conjugate gradient solvers. We point out that Eq. (2) with arbitrary right-hand-side vectors can also be transformed into a system of similar properties utilizing $\mathcal{B}$'s nullspace, which is essentially the connected components spanned by the inductive branches, making it possible to reduce its memory usage as well.

B. Multinode Moment Matching

Moment matching is among the most popular MOR techniques for linear circuit simulations because of its simplicity. It builds a reduced-order model of the circuit by approximating the responses at circuit outputs by a small number of poles and residues computed via matching the actual and the approximated moments. While single-point moment matching techniques like asymptotic waveform evaluation (AWE) [13] compute poles independently for each output, the multinode moment matching (MMM) method [9] assumes the same set of poles across all the outputs, and thus requires much less moments to be matched for the same accuracy.

While MMM is able to handle multiple inputs, we are only interested in the responses under certain initial conditions as required by our proposed framework. Assume that $I(t) = 0$ for $t > 0$ in Eq. (1). Let v_s and i_s be the Laplace transforms of $v(t)$ and $i(t)$ respectively. For a given initial condition of $v(0)$ and $i(0)$, Eq. (1) can be transformed into,

$$A \begin{pmatrix} v_s \\ i_s \end{pmatrix} = \begin{pmatrix} Cv(0) \\ -\mathcal{L}i(0) \end{pmatrix} + s \begin{pmatrix} -Cv_s \\ \mathcal{L}i_s \end{pmatrix}, \tag{5}$$

where the matrix A is defined in Eq. (2). Let the Taylor series of v_s and i_s around $s = 0$ be

$$\begin{aligned} v_s &= \alpha_0 + \alpha_1 s + \alpha_2 s^2 + \alpha_3 s^3 + \cdots, \\ i_s &= \beta_0 + \beta_1 s + \beta_2 s^2 + \beta_3 s^3 + \cdots. \end{aligned} \tag{6}$$

Comparing both sides of Eq. (5) after substituting v_s and i_s by their respective Taylor series, the kth moment (α_k, β_k) for $k = 0, 1, 2, \ldots$ can be computed from given $v(0)$ and $i(0)$ iteratively,

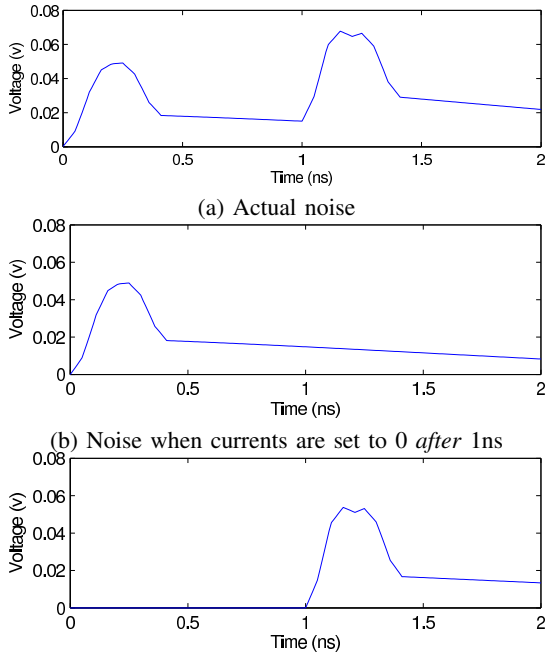
$$A \begin{pmatrix} \alpha_0 \\ \beta_0 \end{pmatrix} = \begin{pmatrix} Cv(0) \\ -\mathcal{L}i(0) \end{pmatrix}, A \begin{pmatrix} \alpha_{k+1} \\ \beta_{k+1} \end{pmatrix} = \begin{pmatrix} -C\alpha_k \\ \mathcal{L}\beta_k \end{pmatrix}. \tag{7}$$

To build a reduced order system of order q, the first $q+1$ moments (α_0, β_0), (α_1, β_1), …, (α_q, β_q) are computed according to Eq. (7). Then, q outputs should be chosen [1] and for every $k = 0, 1, \ldots, q$, a $q \times 1$ moment vector m_k is extracted from (α_k, β_k) corresponding to them. As long as m_1,m_2,…,m_q are linearly independent, the q poles of the reduced order system are computed as the eigenvalues of the matrix Γ defined as,

$$\Gamma \triangleq (m_0 m_1 \cdots m_{q-1})(m_1 m_2 \cdots m_q)^{-1}. \tag{8}$$

Finally, for each output of interests that could be different from the chosen q outputs, its q residues can be obtained by solving a system of linear equations [9]. Overall, since q is always much smaller than $n + l$, MMM is dominated by the computation of the moments in Eq. (7). When the LU decomposition of A is available, e.g. if our proposed framework utilizes a direct solver for time domain simulations, to compute each moment just need a pair of forward and back substitutions, consuming similar running time as a single time step in time domain simulations.

[1] We will discuss how to choose such q outputs in Section III-C.



(a) Actual noise

(b) Noise when currents are set to 0 *after* 1ns

(c) Noise when currents are set to 0 *before* 1ns

Fig. 1. Superposition of Noises: the noise in (a) is the summation of the noises in (b) and (c)

III. The MOR-Assisted Time Domain Simulation Framework

A. Motivation

We demonstrate the motivation of our proposed framework using a node in the ground network from the power grid ibmpg2t in the IBM power grid simulation benchmarks [12]. The actual noise at the node from 0ns to 2ns as shown in Fig. 1 (a) is obtained by a time domain simulation of 100 time steps using a time step of 10ps. Moreover, we show the noise at the node when all the independent current sources are set to 0 *after* 1ns in Fig. 1 (b), and the noise at the node when all the the independent current sources are set to 0 *before* 1ns in Fig. 1 (c). Obviously, according to the superposition property, the actual noise is the summation of the other two.

Since the noises in Fig. 1 (b) and (c) can be obtained independently, it is intuitive to ask whether such problem decomposition leads to efficient parallel and distributed implementations. However, to obtain the noises in Fig. 1 (b) and (c) from time domain simulations will not reduce the wall-clock time. Although to obtain Fig. 1 (c) will require to simulate only half of the time steps since the noise from 0ns to 1ns should be 0, to obtain Fig. 1 (b) will still require to simulate 100 time steps since the "tail" of the noise from 1ns to 2ns remains significant (about 10mv to 20mv).

For practical power grids where any LC loop contains some resistance, the tail of the noises should eventually approach 0 when the independent current sources are set to 0. It is therefore possible to reduce the wall-clock time if there is more time steps to simulate than the time for the tail to become negligible. Nevertheless, it would be more rewarding if one could estimate the tail directly without performing any time domain simulations. Due to the low-pass nature of power grids, the tail behavior is dominated by the poles of the system whose magnitudes are small and whose associated residues, as determined by the state of the system when the inputs become 0, are significant. Therefore, it is actually possible to obtain an accurate estimation of the tail behavior leveraging MOR techniques that are specifically designed to capture such behaviors. Moreover, since we are only interested in the tail behavior when the inputs are 0, we circumvent the difficulties associated with the inputs for MOR techniques when they are applied to circuit simulation problems.

MOR-Assisted Time Domain Simulation

Inputs

T: desired simulation duration.
N: nodes whose noises are of interests.
p: number of available processor cores.
d: additional time domain simulation steps.
q: maximum order of the reduced order system.

Outputs

Noises for the nodes in N at each time step.

Stage 1: Split

1 Split T into p intervals T_1,T_2,. . .,T_p.

Stage 2: Map

2 For each T_j:
3 Obtain $v(t)$ and $i(t)$ during T_j by time domain simulation using the actual currents.
4 Using $v(t)$ and $i(t)$ at the last time step of T_j as the initial condition, simulate d additional time steps assuming all the currents are 0.
5 Using $v(t)$ and $i(t)$ at the dth addtional time step, build a reduced order system of order q for the tail of the noises.
6 For each node in N, output the noise for each time step in T_j and of the d additional time steps, and output the reduced model for its tail.

Stage 3: Reduce

7 For each $x \in N$:
8 Collect the outputs for x from all the tasks in the Map stage.
9 Evaluate the reduced order systems and accumulate the actual noise at x for each time step.

Fig. 2. The Proposed MOR-Assisted Time Domain Simulation Framework

B. The Proposed Framework

Our proposed *MOR-Assisted Time Domain Simulation* framework is shown in Fig. 2. The framework contains three stages Split, Map, and Reduce. The Split stage partitions the desired simulation duration into p intervals that will be handled in the Map stage in parallel. The Map stage includes p tasks, one each for the p intervals. Every task will first apply time domain simulation to obtain a portion of the nodal noises from the actual excitation during the interval, and then apply MOR to estimate the tail of the noises. To allow responses resulting from the poles whose magnitudes are large to die out further, d additional time steps are simulated with zero excitation at the end of each interval. At the end of the Map stage, the noises and the reduced models are emitted as output for all the nodes in the set N whose noises are of interests. The Reduce stage includes one task for each node in N. In each task, the noises and the reduced models for the node are first collected from the outputs of the Map stage, and then evaluated and accumulated to obtain the actual noise.

Our decision to assign one but not more tasks to each processor core in the Map stage is based on the intuition that less tasks will incur less overhead since less reduced order systems are built, and will lead to better accuracy since less tails are estimated. Therefore, the Split stage is essential to balance the loads over multiple processor cores. We believe that it is possible to achieve a good load balancing in practice since the running time for time domain simulations and MOR techniques can be estimated fairly well from the number of time

steps, and the capability of each processor core can be known a priori. Note that such decision also differs our framework from the popular MapReduce framework [4], though our framework may still benefit from MapReduce implementations, e.g. for the communications that distribute the outputs of the Map stage to the Reduce stage.

C. Implementation Details

We choose to perform the time domain simulations with a direct sparse matrix solver. Each machine participating in the computation will compute the LU decompositions for the matrices A and A_h as required by Eq. (2) and (3). The results are then shared among its processor cores. Computations in the Map stage for time domain simulations are then dominated by the pairs of forward and back substitutions.

We choose the multinode moment matching (MMM) technique as discussed in Section II-B to build the reduced order systems. Since the LU decomposition of A is available from time domain simulations, the computations of the moments as outlined in Eq. (7) are also dominated by the pairs of forward and back substitutions. For any given q, to choose q outputs arbitrarily may result in the extracted moment vectors m_1, m_2, ..., m_q being linearly dependent. Actually, there is no guarantee that one can find q outputs for them to be linearly independent – for example, if $v(0) = 0$ and $i(0) = 0$, then all the moments are 0. We propose to address this concern by an algorithm that find the maximum $q' \leq q$ and a set of q' outputs such that the extracted moment vectors m_1, m_2, ..., $m_{q'}$ are linearly independent. Our algorithm starts by forming a $q \times (n+l)$ matrix whose ith row is the ith moment (α_i, β_i). We then perform a partial LU decomposition on this matrix with column pivoting until either the $q' = q$th row is completed or the $q' + 1$th row becomes all 0. The outputs are chosen as the pivots. Note that in our algorithm, the LU decomposition of $(m_1 m_2 \cdots m_{q'})^T$ is computed at the same time. Therefore, we can proceed to compute the matrix Γ according to Eq. (8) immediately.

Our current implementation assumes that all the machines have the same number of processor cores and all the cores have the same capability. Therefore, the Split stage will simply partition T into intervals of equal length. Moveover, for simplicity, we implement the Reduce stage without parallelizing its tasks. We found this stage constributes to only a negligible amount of running time and such sequential implementation is suffice.

D. Complexity Analysis

We provide an analysis of the complexity of our framework based on the choices made in Section III-C. Details follow.

First of all, assume that the maximum number of fill-ins in both the LU decompositions for the matrices A and A_h is m. Usually m is larger than $n + l$ but are much smaller $(n + l)^2$ for sparse systems. Let $|T|$ and $|N|$ be the number of time steps and the number of nodes whose noises are of interests respectively. For the time domain simulation based on a sequential direct solver that serves as the baseline for comparison, the time complexity is $O(m|T|)$ and the space complexity is $O(m + |N||T|)$. Note that if the noises are computed on distributed platforms, then $O(|N||T|)$ data need to be communicated for the noises at $|N|$ nodes for $|T|$ time steps.

In our implementation of the proposed framework, each machine requires an $O(m)$ storage to store the LU decompositions. The Split stage can be completed in $O(p)$ time. In the Map stage, each task will require at most $O(m(\frac{|T|}{p} + d))$ time to perform the time domain simulation and $O(mq)$ time to compute the moments. The rest of the MMM computation is dominated by the partial LU decomposition that requires $O(q^2(n + l))$ time and the residue calculations that requires $O(q^3 + q^2|N|)$ time. Thus the time complexity is $O(q^2(n + l))$ since $|N| \leq n + l$. In summary, each task in the Map stage has a time complexity of $O(m\frac{|T|}{p} + m(d + q) + q^2(n + l))$ and a storage complexity of $O(|N|\frac{|T|}{p} + q(n + l))$. At the end of the Map stage, $O(|T| + pq)$ data need to be communicated for each node in N. In the Reduce stage, each task has a time complexity of $O(pq|T|)$ and a space complexity of $O(|T| + pq)$.

Overall, the wall-clock time of our implementation is dominated by the forward and back substitutions for the time domain simulation and the moments computation in each task of the Map stage. Therefore, in comparison to the baseline, we would expect an ideal speed-up of

$$\text{Speed-up}(p) = \frac{m|T|}{m\frac{|T|}{p} + m(d + q)} = p\frac{1}{1 + \frac{d+q}{|T|/p}}. \quad (9)$$

The overall storage overhead is $O(pq(n + l))$, dominated by the need to store the moments and the residues at each processor core. The overall communication overhead is $O(pq|N|)$ as required for all the residues. Note that our proposed framework needs only two synchronizations, one each between the transition among two consecutive stages, incurring much less overhead in comparison to the time domain simulation via parallel direct and iterative solvers where at least $|T|$ synchronizations are necessary.

IV. Experimental Results

We implement the proposed MOR-Assisted time domain simulation framework in C++ combining the direct sparse matrix solver NICSLU [2] and the MMM techique. Our framework utilizes multiple processor cores on a single machine via POSIX Threads, and multiple machines are coordinated via POSIX sockets. For simplicity, we did not apply parallel LU decompositions and parallel forward and back substitutions with the exception that the LU decompositions of A and A_h are computed simultaneously on two processor cores of each machine. For comparison, we implement a sequential time domain simulation tool utilizing the NICSLU solver. Our code is built by GCC version 4.1 and we run all the experiments on two identical 64-bit Linux servers with dual 2.67GHz Intel X5650 processor (12 processor cores total) and 64GB memory, interconnected on a gigabit Ethernet.

Our experiments are based on the six the IBM power grid simulation benchmarks [12] where every benchmark needs to be simulated for 1000 time steps . For all the benchmarks in each task of the Map stage, there will be $d = 5$ additional time steps and the MMM technique will build a system with at most $q = 20$ poles, which are suffice to capture the tail behaviors as validated by the results. We experiment with multiple configurations of task assignments on the available processor cores on the two servers. The results from a few typical configurations are shown in Table I. In this table, we show the wall-clock time in seconds to complete the 1000 time steps ("t_{1000}") excluding LU decompositions, the overall wall-clock time in seconds ("t_{tot}") including everything from reading the SPICE netlist to writing the output, and the maximum/average errors in uV to the golden solution ("$Err.$"). Both the wall-clock times and the errors for the sequential time domain simulation are similar to those reported by the winners of the recent TAU contest [19], [18], [17].

TABLE I. SPEED-UPS OF MOR-ASSISTED TIME DOMAIN SIMULATIONS OVER SEQUENTIAL SIMULATIONS

	Sequential Simulation			MOR-Assisted Time Domain Simulation Framework							
	1 Core			4 Cores					16 Cores		
	One Server		Err.	One Server		Two Servers		Err.	Two Servers		Err.
name	$t_{1000}(s)$	$t_{tot}(s)$	(uV)	$t_{1000}(s)$	$t_{tot}(s)$	$t_{1000}(s)$	$t_{tot}(s)$	(uV)	$t_{1000}(s)$	$t_{tot}(s)$	(uV)
ibmpg1t	2.30	2.81	53/4	1.00	1.44	1.55	1.99	54/5	0.40	0.84	53/5
ibmpg2t	31.66	37.11	46/4	11.59	15.08	12.21	15.72	46/4	4.86	8.51	46/5
ibmpg3t	85.12	99.76	41/4	31.25	41.86	30.36	41.13	42/4	15.17	26.02	42/4
ibmpg4t	401.07	678.88	131/9	180.18	340.11	154.72	314.16	131/9	85.62	244.93	132/9
ibmpg5t	180.80	215.27	31/3	77.45	99.99	60.58	83.61	35/4	31.81	54.98	38/5
ibmpg6t	264.54	301.31	33/4	99.67	126.46	96.99	123.47	34/4	46.75	72.81	83/6
total	965.49	1335.14		401.14	624.92	356.41	580.07		184.60	408.07	
ratio	1.00	1.00		2.41 X	2.14 X	2.71 X	2.30 X		5.23 X	3.27 X	

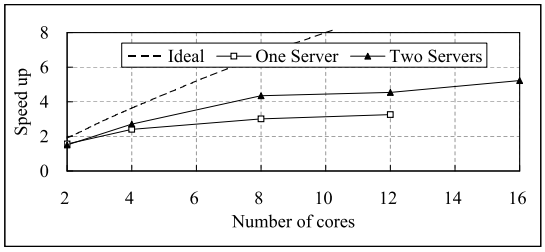


Fig. 3. Speed-ups under Different Configurations

Our proposed framework is able to achieve more than $2X$ speed-ups utilizing only 4 cores either on the same server or the two servers without losing accuracy, which is comparable to the best speed-up that can be achieved for those aforementioned TAU contest winners. When 16 cores are used, 8 from each server, our proposed framework is able to achieve more than $5X$ speed-ups to complete the 1000 time steps. The speed-ups of the overall wall-clock times in such case are limited to a little bit more than $3X$ due to the LU decompostions. This may or may not be a concern for even larger power grids since an iterative solver must be used where setting up a preconditioner may consume far less time relatively than LU decompostions. Note that when more cores are used on these two machines, we have not observed further speed-ups, possibly because the ratio of the overhead to the necessary simulation steps, roughly $\frac{p}{40}$, increases considerably in such cases.

It is important to point out that our framework actually performs better with two servers than a single one for the same number of cores. As shown in Fig. 3, the speed-ups on the distributed platform are consistently better starting from 4 cores, though on both platforms the speed-ups are far from the ideal one $p\frac{1}{1+\frac{p}{40}}$ as predicted in Eq. (9). We believe that this is due to the fact that forward and back substitutions will incur considerable overhead in memory traffic – though all the tasks in the Map stage are independent of each other, memory access contentions exist and cores have to wait until data become available. The overhead is so large that it is more rewarding to "pay" communication cost in order to "buy" memory bandwidth from other machines. Note that although similar limitations have been observed in [17], distributing forward and back substitutions to multiple machines might not be rewarding since the communication cost there is much larger than in our proposed framework.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed the MOR-assisted time domain simulation framework that combines time domain simulations and MOR techniques for scalable power grid transient analysis. Experiments showed that by utilizing a direct sparse matrix solver and the multinode moment matching technique, more than 5X speed-up over the sequential simulation were achieved to complete 1000 time steps on 16 processor cores distributed over two servers for the IBM power grid simulation benchmarks. The remaining questions we plan to address in the future are that to what extent one can trust the output when the golden is not available, that whether it is possible to improve single machine performance by coordinating memory accesses, and that how the framework can be adapted in a distributed environment where machines with different capabilities exist.

REFERENCES

[1] T.-H. Chen and C. C.-P. Chen. Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods. In *DAC*, pages 559–562, 2001.

[2] X. Chen, W. Wu, Y. Wang, H. Yu, , and H. Yang. EScheduler-based data dependence analysis and task scheduling for parallel circuit simulation. *IEEE Trans. Circuits and Systems II: Express Briefs*, 58(10):702–706, Oct. 2011.

[3] K. Daloukas, N. Evmorfopoulos, G. Drasidis, M. Tsiampas, P. Tsompanopoulou, and G. I. Stamoulis. Fast transform-based preconditioners for large-scale power grid analysis on massively parallel architectures. In *ICCAD*, pages 384–391, 2012.

[4] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, Jan. 2008.

[5] Z. Feng and P. Li. Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms. In *ICCAD*, pages 647–654, 2008.

[6] Z. Feng, X. Zhao, and Z. Zeng. Robust parallel preconditioned power grid simulation on GPU with adaptive runtime performance modeling and optimization. *IEEE TCAD*, 30(4):562–573, Apr. 2011.

[7] C.-W. Ho, A. Ruehli, and P. Brennan. The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems*, 22(6):504–509, June 1975.

[8] X. Hu, W. Zhao, P. Du, A. Shayan, and C.-K. Cheng. An adaptive parallel flow for power distribution network simulation using discrete fourier transform. In *ASPDAC*, pages 125–130, 2010.

[9] Y. I. Ismail. Improved model-order reduction by using spacial information in moments. *IEEE TVLSI*, 11(5):900–908, Oct. 2003.

[10] Y.-M. Lee, Y. Cao, T.-H. Chen, J. Wang, and C.-P. Chen. HiPRIME: hierarchical and passivity preserved interconnect macromodeling engine for RLKC power delivery. *IEEE TCAD*, 24(6):797–806, June 2005.

[11] D. Li, S.-D. Tan, and B. McGaughy. ETBR: Extended truncated balanced realization method for on-chip power grid network analysis. In *DATE*, pages 432–437, 2008.

[12] Z. Li, R. Balasubramanian, F. Liu, and S. Nassif. 2012 TAU power grid simulation contest: benchmark suite and results. In *ICCAD*, pages 643–646, 2012.

[13] L. T. Pillage and R. A. Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE TCAD*, 9(4):352–366, Apr. 1990.

[14] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen. Parallel domain decomposition for simulation of large-scale power grids. In *ICCAD*, pages 54–59, 2007.

[15] J. Wang and T. Nguyen. Extended Krylov subspace method for reduced order analysis of linear circuits with multiple sources. In *DAC*, pages 247–252, 2000.

[16] S.-H. Weng, Q. Chen, N. Wong, and C.-K. Cheng. Circuit simulation via matrix exponential method for stiffness handling and parallel processing. In *ICCAD*, pages 407–414, 2012.

[17] X. Xiong and J. Wang. Parallel forward and back substitution for efficient power grid simulation. In *ICCAD*, pages 660–663, 2012.

[18] J. Yang, Z. Li, Y. Cai, and Q. Zhou. PowerRush: efficient transient simulation for power grid analysis. In *ICCAD*, pages 653–659, 2012.

[19] T. Yu and M. D. F. Wong. PGT_SOLVER: an efficient solver for power grid transient analysis. In *ICCAD*, pages 647–652, 2012.

[20] T. Yu, Z. Xiao, and M. D. F. Wong. Efficient parallel power grid analysis via additive schwarz method. In *ICCAD*, pages 399–406, 2012.

[21] M. Zhao, R. Panda, S. Sapatnekar, and D. Blaauw. Hierarchical analysis of power distribution networks. *IEEE TCAD*, 21(2):159–168, Feb. 2002.

[22] S. Zhao, K. Roy, and C.-K. Koh. Frequency domain analysis of switching noise on power supply network. In *ICCAD*, pages 487–492, 2000.