MATEX: A Distributed Framework for Transient Simulation of Power Distribution Networks

Hao Zhuang[§], Shih-Hung Weng[†], Jeng-Hau Lin[§], Chung-Kuan Cheng[§]

[§]Computer Science & Engineering Department, University of California, San Diego, CA, 92093
[†]Facebook Inc., Menlo Park, CA, 94025
zhuangh@ucsd.edu, shweng@fb.com, jel252@ucsd.edu, ckcheng@ucsd.edu

ABSTRACT

We proposed *MATEX*, a distributed framework for transient simulation of power distribution networks (PDNs). MATEX utilizes matrix exponential kernel with Krylov subspace approximations to solve differential equations of linear circuit. First, the whole simulation task is divided into subtasks based on decompositions of current sources, in order to reduce the computational overheads. Then these subtasks are distributed to different computing nodes and processed in parallel. Within each node, after the matrix factorization at the beginning of simulation, the adaptive time stepping solver is performed without extra matrix re-factorizations. MATEX overcomes the stiffness hinder of previous matrix exponential-based circuit simulator by *rational* Krylov subspace method, which leads to larger step sizes with smaller dimensions of Krylov subspace bases and highly accelerates the whole computation. MATEX outperforms both traditional fixed and adaptive time stepping methods, e.g., achieving around 13X over the trapezoidal framework with fixed time step for the IBM power grid benchmarks.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids, Simulation. J.6 [Computer-aided engineering]: Computer-aided design (CAD).

General Terms

Algorithms, Design, Theory, Verification, Performance

Keywords

Circuit Simulation, Power Distribution Networks, Power Grid, Transient Simulation, Matrix Exponential, Krylov Subspace, Distributed Computing, Parallel Processing.

1. INTRODUCTION

Modern VLSI design verification relies heavily on the analysis of power distribution network (PDN) to estimate power supply noises. PDN is often modeled as a large-scale linear circuit with voltage supplies and time-varying current sources [8, 21]. Such circuit is extremely large, which makes the corresponding transient simulation very time-consuming. Therefore, scalable and theoretically elegant algorithms for the transient simulation of linear circuits have been always favored. Nowadays, the emerging multi-core, many-core platforms bring powerful computing resource and opportunities for parallel computing. Even more, cloud computing techniques [1] drive distributed systems scaling to thousands of computing nodes [6], etc. Such distributed systems will be also promising computing resources in EDA industry. However, building scalable and efficient distributed algorithmic framework for transient linear circuit simulation framework is still a challenge to leverage these powerful computing tools. Previous works [7, 14–16] have been made in order to improve circuit simulation by novel algorithms, parallel processing and distributed computing.

Traditional numerical methods solve differential algebra equations (DAEs) explicitly, e.g., forward Euler, or implicitly, e.g., backward Euler (BE), trapezoidal (TR) method, which are based on low order polynomial approximations. Due to the stiffness of systems, which comes from a wide range of time constants of a circuit, the explicit methods require small time step sizes to ensure the stability. In contrast, implicit methods can deal with this problem because of their larger stability regions. However, at each time step, these methods have to solve a linear system, which is sparse and often ill-conditioned. Due to the requirement of a robust solution, compared to iterative matrix solvers [12], direct matrix solvers [5] are often favored for VLSI circuit simulation, and thus adopted by state-of-the-art power grid (PG) solvers in TAU PG simulation contest [18–20]. During transient simulation, these solvers require only one matrix factorization (LU or Cholesky factorization) at the beginning. Then, the following transient computation, at each fixed time step, needs only a pair of forward and backward substitutions, which achieves better efficiency over adaptive stepping methods by reusing the factorized matrix [8, 18, 20].

Beyond traditional methods, a new class of methods called exponential time differencing (ETD) has been embraced by MEXP [15]. The major complexity of ETD is caused by matrix exponential computations. MEXP utilizes standard Krylov subspace method based on [11] to approximate matrix exponential and vector product. MEXP can solve the DAEs with high polynomial approximations [11, 15] than traditional ones. Another merit of using MEXP-like SPICE simulation for linear circuit is the adaptive time stepping, which can proceed without re-factorizing matrices on-the-fly, while the traditional counterparts cannot avoid such time-consuming process during the adaptive time marching. Nevertheless, when simulating stiff circuits, utilizing standard Krylov subspace method requires large dimension of basis in order to preserve the accuracy of MEXP approximation. It may pose memory bottleneck and degrade the adaptive stepping performance of MEXP.

In this paper, we propose a distributed algorithmic framework for PDN transient simulation, called as *MATEX*, which inherits the matrix exponential kernel. First, the PDN's input sources are partitioned into groups based on their similarity. They are assigned to different computing nodes to run the corresponding PDN transient simulations. Then, the results among nodes are summed up, according to the well-known superposition property of linear system. This partition reduces the chances of generating Krylov subspaces and enlarges the time periods of reusing them during the transient simulation at each node, which brings huge computational advantage. In addition, we also highly accelerate the circuit solver by adopting *inverted* and *rational* Krylov subspace methods for the computation of matrix exponential and vector product. We find the rational

DAC '14, June 01-05 2014, San Francisco, CA, USA

http://dx.doi.org/10.1145/2593069.2593160.

Krylov subspace method is the most efficient one, which helps MATEX leverage its flexible adaptive time stepping by reusing factorized matrix at the beginning of transient simulation. In IBM power grid simulation benchmarks, our framework gains around **13X** speedup on average in transient computing part after its matrix factorization, compared to the commonly adopted TR method with fixed time step. The overall speedup is **7X**.

Paper Organization. Section 2 introduces the background of linear circuit simulation and matrix exponential formulations. Section 3 presents overall framework of MATEX. Section 4 shows numerical results and Section 5 concludes this paper.

2. PRELIMINARIES

2.1 Transient Simulation of Linear Circuits

Transient linear circuit simulation is the foundation of PDN simulation. It is formulated as DAEs via modified nodal analysis (MNA),

$$\mathbf{C}\dot{\mathbf{x}}(t) = -\mathbf{G}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \tag{1}$$

where $\mathbf{C}$ is the matrix resulting from capacitive and inductive elements. $\mathbf{G}$ is the conductive matrix, and $\mathbf{B}$ is the input selector matrix. $\mathbf{x}(t)$ is the vector of time-varying node voltages and branch currents. $\mathbf{u}(t)$ is the vector of supply voltage and current sources. In PDN, such current sources are often characterized as pulse inputs [8, 10]. To solve Eq. (1) numerically, it is, commonly, discretized with time step h and transformed to a linear algebraic system. Given an initial condition $\mathbf{x}(0)$ from DC analysis, or previous time step $\mathbf{x}(t)$, For a time step h, $\mathbf{x}(t+h)$ can be obtained by traditional *low order approximation* methods, e.g., TR, which is an implicit second-order method, and probably most commonly used strategy for large scale circuit simulation.

$$(\frac{\mathbf{C}}{h} + \frac{\mathbf{G}}{2})\mathbf{x}(t+h) = (\frac{\mathbf{C}}{h} - \frac{\mathbf{G}}{2})\mathbf{x}(t) + \mathbf{B}\frac{(\mathbf{u}(t) + \mathbf{u}(t+h))}{2} \tag{2}$$

Besides, TR with fixed time step h is an efficient framework and adopted by the top PG solvers in 2012 TAU PG simulation contest [8, 18–20].

2.2 Exponential Time Differencing Method

The solution of Eq. (1) can be obtained analytically [4]. For simple illustration, we convert Eq. (1) into

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{b}(t), \tag{3}$$

when $\mathbf{C}$ is not singular, $\mathbf{A} = -\mathbf{C}^{-1}\mathbf{G}$ and $\mathbf{b}(t) = \mathbf{C}^{-1}\mathbf{B}\mathbf{u}(t)$. Given the solution at time t and a time step h, the solution at $t+h$ is

$$\mathbf{x}(t+h) = e^{h\mathbf{A}}\mathbf{x}(t) + \int_0^h e^{(h-\tau)\mathbf{A}}\mathbf{b}(t+\tau)d\tau. \tag{4}$$

Assuming that the input $\mathbf{u}(t)$ is piecewise linear (PWL), e.g. $\mathbf{u}(t)$ is linear within every time step, we can integrate the last term of Eq. (4), analytically, turning the solution with matrix exponential operator:

$$\begin{aligned}\mathbf{x}(t+h) &= e^{h\mathbf{A}}(\mathbf{x}(t) + \mathbf{A}^{-1}\mathbf{b}(t) + \mathbf{A}^{-2}\frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h}) \\ &\quad -(\mathbf{A}^{-1}\mathbf{b}(t+h) + \mathbf{A}^{-2}\frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h})\end{aligned} \tag{5}$$

For the time step choice, *input transition spots (TS)* refer to the time points where slopes of input sources vector changes. Therefore, for Eq. (5), the maximum time step starting from t is $(t_s - t)$, where t_s is the smallest one in TS larger than t.

In Eq. (5), $\mathbf{A}$ in $e^{\mathbf{A}}\mathbf{v}$ is usually above millions, making the direct computation infeasible.

2.3 Matrix Exponential Computation by Standard Krylov Subspace Method

The complexity of $e^{\mathbf{A}}\mathbf{v}$ can be reduced using Krylov subspace method and still maintained in a high order polynomial approximation [11], which has been deployed by MEXP [15]. In this paper, we call the Krylov subspace utilized in MEXP as *standard Krylov subspace*, due to its straightforward usage of $\mathbf{A}$ when generating basis through Arnoldi process in Alg. 1. First, we reformulate Eq. (5) into

$$\mathbf{x}(t+h) = e^{h\mathbf{A}}(\mathbf{x}(t) + \mathbf{F}(t,h)) - \mathbf{P}(t,h) \tag{6}$$

where $\mathbf{F}(t,h) = \mathbf{A}^{-1}\mathbf{b}(t) + \mathbf{A}^{-2}\frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h}$ and $\mathbf{P}(t,h) = (\mathbf{A}^{-1}\mathbf{b}(t+h) + \mathbf{A}^{-2}\frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h})$. The standard Krylov subspace $\mathrm{K_m}(\mathbf{A}, v) := \mathrm{span}\{\mathbf{v}, \mathbf{A}\mathbf{v}, \cdots, \mathbf{A}^{m-1}\mathbf{v}\}$ obtained by Arnoldi process has the relation $\mathbf{A}\mathbf{V_m} = \mathbf{V}_m\mathbf{H}_m + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^{\mathsf{T}}$, where $h_{m+1,m}$ is the $(m+1, m)$ entry of Hessenberg matrix $\mathbf{H}_m$, and e_m is the m-th unit vector. The matrix exponential and vector product is computed via $e^{h\mathbf{A}}\mathbf{v} \approx \|\mathbf{v}\|\mathbf{V}_m e^{h\mathbf{H}_m}\mathbf{e}_1$. The $\mathbf{H}_m$ is usually much smaller compared to $\mathbf{A}$. The posterior error term is

$$\|r_m(h)\| = \|\mathbf{v}\|\left|h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^{\mathbf{T}}e^{h\mathbf{H}_m}\mathbf{e}_1\right| \tag{7}$$

To generate $\mathbf{x}(t+h)$ by Alg. 1, we use $[\mathbf{L}, \mathbf{U}] = \mathrm{LU\_Decompose}(\mathbf{X}_1)$, where, for standard Krylov subspace, $\mathbf{X}_1 = \mathbf{C}$, and $\mathbf{X}_2 = \mathbf{G}$ as inputs. The error budget ϵ and Eq. (7) are used to determine the convergence condition in current time step h with an order j of Krylov subspace dimension for $e^{\mathbf{A}}\mathbf{v}$ approximation (from line 10 to line 12).

Algorithm 1: MATEX_Arnoldi

Input: $\mathbf{L}, \mathbf{U}, \mathbf{X}_2, h, t, \mathbf{x}(t), \epsilon, \mathbf{P}(t,h), \mathbf{F}(t,h)$
Output: $\mathbf{x}(t+h), \mathbf{V}_m, \mathbf{H}_m, \mathbf{v}$

1 $\mathbf{v} = \mathbf{x}(t) + \mathbf{F}(t,h), \mathbf{v}_1 = \frac{\mathbf{v}}{\|\mathbf{v}\|}$;
2 **for** $j = 1 : m$ **do**
3 $\quad \mathbf{w} = \mathbf{U}\backslash(\mathbf{L}\backslash(\mathbf{X}_2\mathbf{v}_j))$; `/* a pair of forward and backward substitutions */`
4 $\quad$ **for** $i = 1 : j$ **do**
5 $\quad\quad h_{i,j} = \mathbf{w}^T\mathbf{v}_i$;
6 $\quad\quad \mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$;
7 $\quad$ **end**
8 $\quad h_{j+1,j} = \|\mathbf{w}\|$;
9 $\quad \mathbf{v}_{j+1} = \frac{\mathbf{w}}{h_{j+1,j}}$;
10 $\quad$ **if** $||\mathbf{r}_j(h)|| < \epsilon$ **then**
11 $\quad\quad m = j$; break;
12 $\quad$ **end**
13 **end**
14 $\mathbf{x}(t+h) = \|\mathbf{v}\|\mathbf{V_m}e^{h\mathbf{H}_m}\mathbf{e}_1 - \mathbf{P}(t,h)$;

2.4 Discussions of MEXP

The input term $\mathbf{b}$ embedded in Eq. (4) serves a double-edged sword in MEXP. First, the flexible time stepping can choose any time spot until the next input transition spot t_s, as long as the approximation of $e^{\mathbf{A}}\mathbf{v}$ is accurate enough. The Krylov subspace can be reused when $t + h \in [t, t_s]$, only by scaling $\mathbf{H}_m$ with h in $\mathbf{x}(t+h) = \|\mathbf{v}\|\mathbf{V_m}e^{h\mathbf{H}_m}\mathbf{e}_1 - \mathbf{P}(t,h)$. This is an important feature that even doing the adaptive time stepping, we can still use the last Krylov subspaces.

However, the region before the next transition t_s may be shortened when there are a lot of independent input sources injected into the linear system. It leads to more chances of generating new Krylov subspace. This issue is addressed in Sec. 3.1 and Sec. 3.2.

The standard Krylov subspace may not be computationally efficient when simulating stiff circuits based on MEXP[15, 16]. For the accuracy of approximation of $e^{\mathbf{A}}\mathbf{v}$, large dimension of Krylov subspace basis is required, which not only brings the computational complexity but also consumes huge memory. Besides, for a circuit with singular $\mathbf{C}$, during the generation of standard Krylov subspace, a *regularization* process is required to convert such $\mathbf{C}$ into non-singular one, which is time-consuming for large scale circuits. These two problems are solved in Sec. 3.3.

3. MATEX FRAMEWORK

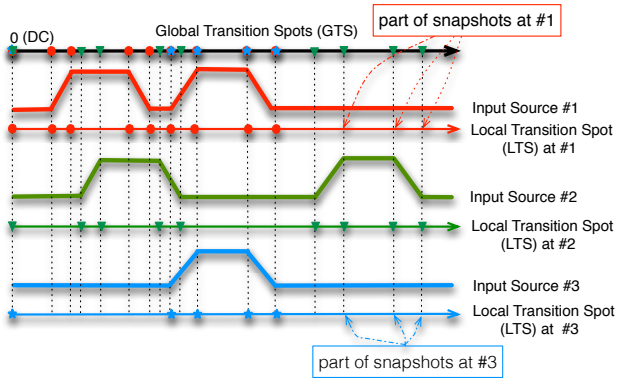

Figure 1: Illustration of input transitions. GTS: Global Transition Spots; LTS: Local Transition Spots; Snapshots: the crossing positions by dash lines and LTS $\#k$ without solid points.

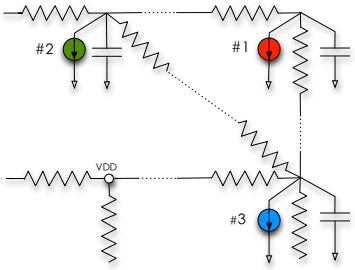

Figure 2: Part of a PDN model with input sources from Fig. 1

3.1 Motivation

Matrix exponential kernel with Kyrlov subspace method can solve Eq. (1) with larger time steps than lower order approximation methods. Our motivation is to leverage this advantage to reduce the number of time step and accelerate the transient simulation. However, there are usually many input currents in PDNs, which narrow the regions for the time stepping of matrix exponential-based method. We want to utilize the well-known superposition property of linear system and distributed computing model to tackle this challenge. To illustrate our framework briefly, we first define three terms:

Definition: *Local Transition Spot* (LTS) is the set of TS at an input source to the PDN.

Definition: *Global Transition Spot* (GTS) is the union of LTS among all the input sources to the PDN.

Definition: *Snapshot* denotes a set $GTS \setminus LTS$ at an input source.

If we simulate the PDN with respective to all the inputs, GTS are the places where generations of Krylov subspace cannot be avoided. For example, there are three input sources in a PDN (Fig. 2). The input waveforms are shown in Fig. 1. Then, the first line is that GTS, which is contributed by all LTS from input sources #1, #2 and #3.

However, we can partition the task to sub-tasks by simulating each input sources individually. Then, each sub-task only needs to generate Krylov subspaces based on its own LTS and keep track of *Snapshot* for the later usage of summation via superposition. In addition, the points in *Snapshot* between two points $l_1, l_2 \in LTS$ ($l_1 < l_2$), can reuse the Krylov subspace generated at l_1, which is mentioned in Sec. 2.4. For each node, the chances of Krylov subspaces generations are reduced and the time periods of reusing these subspaces are enlarged locally, which bring huge computational benefit when processing these subtasks in parallel.

Above, we divide the simulation task by input sources. We can, more aggressively, decompose the task according to the

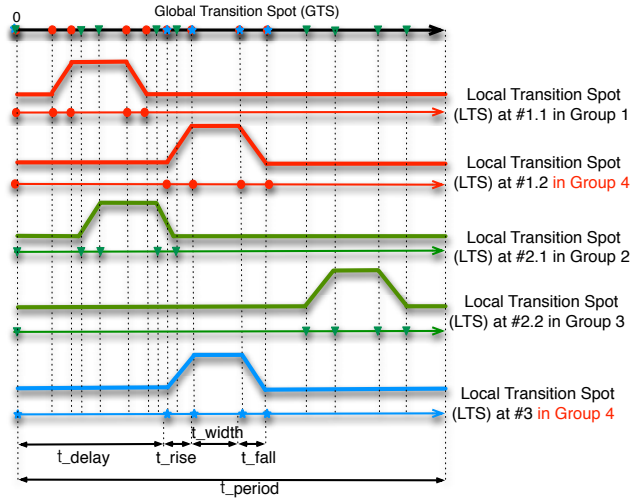

Figure 3: Grouping of "bump" shape transitions for sub-task simulation. The matrix exponential-based method can utilize adaptive stepping in each LTS and reuse Krylov subspace generated at the most recent solid point. However, traditional methods (TR, BE, etc) still need to do time marching step by step, either by pairs of forward and backward substitutions to proceed with fixed time step, or re-factorizing matrix and solving linear system when using adaptive stepping strategy. (Pulse input information: t_delay: initial delay time; t_rise: rise time; t_width: width of pulse-wise; t_fall: fall time; t_period: period).

"bump" shapes within such input pulse sources. We group the ones which have the same (t_delay, t_rise, t_fall, t_width) into one set, which is shown in Fig. 3. There are 4 groups in Fig. 3, Group 1 contains LTS#1.1, Group 2 contains LTS#2.1, Group 3 contains LTS #2.2, and Group 4 contains LTS #1.2 and #3.

3.2 MATEX Framework

Our proposed framework MATEX is shown in Fig. 4. After pre-computing GTS and decomposing LTS based on "bump" shape (Fig. 3), we group them and form LTS $\#1 \sim \#K$ (*Note: there are alternative decomposition strategies. It is also easy to extend the work to deal with different input waveforms. We try to keep this part as simple as possible to emphasize our framework*).

MATEX scheduler sends out GTS and LTS to different MATEX slave node. Then the simulations are processed in parallel. There are no communications among nodes before the "write back". Within each slave node, "circuit solver" (Alg. 2) computes transient response with varied time steps. Solutions are obtained without re-factorizing matrix during the transient computing. After finishing all simulations from slave nodes, they writes back the results and informs the MATEX scheduler.

3.3 Circuit Solver Accelerations

As mention in Sec. 2.4, standard Krylov subspace approximation in MEXP [15] is not computationally efficient for stiff circuit. The reason is that Hessenberg matrix $\mathbf{H}_m$ of standard Krylov subspace tends to approximate the large magnitude eigenvalues of $\mathbf{A}$ [13]. Due to the exponential decay of higher order terms in Taylor's expansion, such components are not the crux of circuit system's behavior [2, 13]. Dealing with stiff circuit, therefore, needs to gather more vectors into subspace basis and increase the size of $\mathbf{H}_m$ to fetch more useful components, which results to both memory overhead and computational complexity into Krylov subspace generations during time stepping. Direct deploying MEXP into MATEX's *Circuit Solver* is not efficient to leverage the benefits of local flexible and larger time stepping. In the following subsections, we adopt the idea from *spectral transformation* [2, 13] to effectively capture small magnitude eigenvalues in $\mathbf{A}$, leading to a fast yet accurate

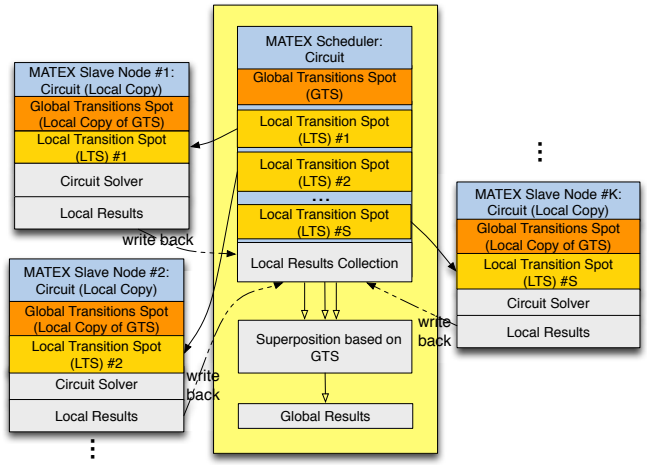

Figure 4: The flow of MATEX framework

Algorithm 2: MATEX Circuit Solver Algorithm

Input: $LTS\ \#k$, GTS, $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{P}_k$, $\mathbf{F}_k$, which contain the corresponding $\mathbf{b}$ for node k.
Output: Local solution $\mathbf{x}$ along GTS in node $k \in [1, \cdots, S]$, where S is the number of nodes

1 $t = T_{start}$;
2 $\mathbf{x}(t)$ = Local_Initial_Solution;
3 $[\mathbf{L}, \mathbf{U}]$ = LU_Decompose($\mathbf{X}_1$);
4 **while** $t \leq T_{end}$ **do**
5 Compute maximum allowed step size h based on GTS;
6 **if** $t \in LTS\ \#k$ **then**
  `/* Generate the Krylov subspace for the time point of LTS and compute x */`
7 $[\mathbf{x}(t+h), \mathbf{V}_m, \mathbf{H}_m, \mathbf{v}] =$ MATEX_Arnoldi($\mathbf{L}, \mathbf{U}, \mathbf{X}_2, h, t, \mathbf{x}(t), \epsilon, \mathbf{P}_k(t,h), \mathbf{F}_k(t,h)$);
  $a_{lts} = t$;
8 **end**
9 **else**
  `/* Compute the x at Snapshot by reusing the latest Krylov subspace */`
10 $h_a = t + h - a_{lts}$;
11 $\mathbf{x}(t+h) = \|\mathbf{v}\|\mathbf{V_m} e^{h_a \mathbf{H}_m} \mathbf{e}_1 - \mathbf{P}_k(t,h)$;
12 **end**
13 $t = t + h$;
14 **end**

Circuit Solver for MATEX.

3.3.1 Matrix Exponential and Vector Computation by Inverted Krylov Subspace (I-MATEX)

Instead of $\mathbf{A}$, we use $\mathbf{A}^{-1}$ (or -$\mathbf{G}^{-1}\mathbf{C}$) as our target matrix to form $\mathrm{K_m}(\mathbf{A}^{-1}, v) := \mathrm{span}\{\mathbf{v}, \mathbf{A}^{-1}\mathbf{v}, \cdots, \mathbf{A}^{-(m-1)}\mathbf{v}\}$. Intuitively, by inverting $\mathbf{A}$, the small magnitude eigenvalues become the large ones of $\mathbf{A}^{-1}$. The resulting $\mathbf{H}'_m$ is likely to capture these eigenvalues first. Based on Arnoldi algorithm, the inverted Krylov subspace has the relation $\mathbf{A}^{-1}\mathbf{V_m} = \mathbf{V}_m\mathbf{H}'_m + h'_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^{\mathrm{T}}$. The matrix exponential $e^{\mathbf{A}}\mathbf{v}$ is calculated as $\|\mathbf{v}\|\mathbf{V}_m e^{h\mathbf{H'}_m^{-1}}\mathbf{e}_1$. To put this method into Alg. 1, just by modifying the input variables, $\mathbf{X}_1 = \mathbf{G}$ for the LU decomposition, and $\mathbf{X}_2 = \mathbf{C}$. In the line 14 of Alg. 1, $\mathbf{H}_m = \mathbf{H'}_m^{-1}$. The posterior error approximation is

$$\|\mathbf{r}_m(h)\| = \|\mathbf{v}\| \left| \mathbf{A} h'_{m+1,m} \mathbf{v}_{m+1} \mathbf{e}_m^T \mathbf{H'}_m^{-1} e^{h\mathbf{H'}_m^{-1}} \mathbf{e}_1 \right| \quad (8)$$

which is derived from residual-based error approximation in [2].

3.3.2 Matrix Exponential and Vector Computation by Rational Krylov Subspace (R-MATEX)

The shift-and-invert Krylov subspace basis [13] is designed to confine the spectrum of $\mathbf{A}$. Then, we generate Krylov subspace via $\mathbf{K_m}((\mathbf{I}-\gamma\mathbf{A})^{-1}, \mathbf{v}) = span\{\mathbf{v}, (\mathbf{I}-\gamma\mathbf{A})^{-1}\mathbf{v}, \cdots, (\mathbf{I}-\gamma\mathbf{A})^{-(m-1)}\mathbf{v}\}$, where γ is a predefined parameter. With this shift, all the eigenvalues' magnitudes are larger than one. Then the invert limits the magnitudes smaller than one. According to [2,13], the shift-and-invert basis for matrix exponential-based transient simulation is not very sensitive to γ, once it is set to around the order near time steps used in transient simulation. The similar idea has been applied to simple power grid simulation with matrix exponential method [22]. Here, we generalize this technique and integrate into MATEX. The Arnoldi process constructs $\mathbf{V_m}$ and $\mathbf{H_m}$, and the relationship is given by $(\mathbf{I}-\gamma\mathbf{A})^{-1}\mathbf{V_m} = \mathbf{V_m}\widetilde{\mathbf{H}}_\mathbf{m} + \tilde{h}_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^T$, we can project the $e^{\mathbf{A}}$ onto the rational Krylov subspace as follows.

$$e^{\mathbf{A}h}\mathbf{v} \approx \|\mathbf{v}\|\,\mathbf{V_m} e^{h\mathbf{H}_m} e_1, \quad (9)$$

where $\mathbf{H_m} = \frac{\mathbf{I}-\widetilde{\mathbf{H}}_m^{-1}}{\gamma}$ for the line 14 of Alg. 1. Following the same procedure [2], posterior error approximation is derived as

$$\|\mathbf{r}_m(h)\| = \|\mathbf{v}\| \left| \frac{\mathbf{I}-\gamma\mathbf{A}}{\gamma} \tilde{h}_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^T \widetilde{\mathbf{H}}_m^{-1} e^{h\mathbf{H}_m}\mathbf{e}_1 \right| \quad (10)$$

Note that in practice, instead of computing $(\mathbf{I}-\gamma\mathbf{A})^{-1}$ directly, $(\mathbf{C}+\gamma\mathbf{G})^{-1}\mathbf{C}$ is utilized. The corresponding Arnoldi process shares the same skeleton of Alg. 1 and Alg. 2 with input matrices $\mathbf{X}_1 = (\mathbf{C}+\gamma\mathbf{G})$ for the LU decomposition, and $\mathbf{X}_2 = \mathbf{C}$.

3.3.3 Regularization-Free Matrix Exponential Method

When dealing singular $\mathbf{C}$, MEXP needs the regularization process [3] to remove the singularity of DAE in Eq. (1). It is because MEXP is required to factorize $\mathbf{C}$ in Alg. 1. This brings extra computational overhead when the case is large. Actually, it is not necessary if we can obtain the generalized eigenvalues and corresponding eigenvectors for matrix pencil $(-\mathbf{G}, \mathbf{C})$. Based on [17], we derive the following lemma,

LEMMA 1. *Considering a homogeneous system* $\mathbf{C}\dot{\mathbf{x}} = -\mathbf{G}\mathbf{x}$, $\mathbf{u}$ *and* λ *are the eigenvector and eigenvalue of matrix pencil* $(-\mathbf{G}, \mathbf{C})$, *then* $\mathbf{x} = e^{t\lambda}\mathbf{u}$ *is a solution of this system.*

An important observation is that we can remove such regularization process out of MATEX, because during Krylov subspace generation, there is no need of computing $\mathbf{C}^{-1}$ explicitly. Instead, we factorize $\mathbf{G}$ for inverted Krylov subspace basis generation (I-MATEX), or $(\mathbf{C}+\gamma\mathbf{G})$ for rational Krylov basis (R-MATEX). Besides, $\mathbf{H}'_m$ and $\widetilde{\mathbf{H}}_\mathbf{m}$ are invertible, which contain corresponding important generalized eigenvalues/eigenvectors from matrix pencil $(-\mathbf{G}, \mathbf{C})$, and define the behavior of linear dynamic system in Eq. (1).

In term of error estimation, because $\mathbf{C}$ is singular, $\mathbf{A}$ cannot be formed explicitly. However, for a certain lower bound of basis number, these two Krylov subspace methods begin to converge, and the error of matrix exponential approximation is reduced quickly. Empirically, the estimation can be replaced with $\|r_m(h)\| = \|\mathbf{v}\| \left| h_{m+1,m}\mathbf{e}_m^{\mathbf{T}} e^{h\mathbf{H}_m}\mathbf{e}_1 \right|$ to approximate Eq. (8), where $\mathbf{H}_m = \mathbf{H'}_m^{-1}, h_{m+1,m} = h'_{m+1,m}$ for inverted Krylov method; $\mathbf{H}_m = \frac{\mathbf{I}-\widetilde{\mathbf{H}}_m^{-1}}{\gamma}, h_{m+1,m} = \tilde{h}_{m+1,m}$ for rational Krylov method.

Note that the larger step R-MATEX utilizes, the smaller error it will have. Fig. 5 shows, when time step h increases, the error between accurate solution and Krylov based approximation $|e^{h\mathbf{A}}\mathbf{v} - \mathbf{V}_m e^{h\mathbf{H}_m}\mathbf{e}_1|$ is reduced. It is because the large step, the more dominating role first smallest magnitude eigenvalues play, which are well captured by rational Krylov subspace-based method [13]. In our MATEX, this property is very crucial factor for large time stepping. Therefore, once we obtain an accurate enough solution and Krylov subspace in line 7 of Alg. 2, we can reuse them in line 14.

3.4 Complexity Analysis

Suppose on average we have Krylov subspace basis dimension m at each time step along the time span, one pair of forward and

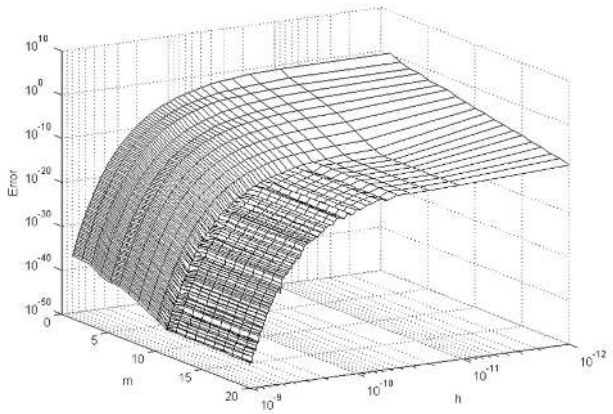

Figure 5: $|e^{h\mathbf{A}}\mathbf{v} - \mathbf{V}_m e^{h\mathbf{H}_m}\mathbf{e}_1|$ **vs. time step** h **and dimension of rational Krylov subspace basis (m).** $\mathbf{H}_m = \frac{\mathbf{I}-\widetilde{\mathbf{H}}_m^{-1}}{\gamma}$; γ **is fixed; A is a relative small matrix and computed by MATLAB** *expm* **function; Therefore,** $e^{h\mathbf{A}}\mathbf{v}$ **serves as the baseline for accuracy. It is observed that error reduces when** h **increases.**

backward substitutions has time complexity T_{bs}. The matrix exponential evaluation using $\mathbf{H}_m$ is T_H which costs time complexity $O(m^3)$, plus extra T_e to form $\mathbf{x}$, which costs $O(nm^2)$. The total time complexity of other serial parts is T_{serial}, which includes matrix factorizations, etc. Given K points of GTS, without decomposition of input transitions, the time complexity is $KmT_{bs} + K(T_H + T_e) + T_{serial}$. After dividing the input transitions and send to enough computing nodes, we have k points of LTS for each node based on the input feature extraction and grouping (e.g., $k = 5$ for one "bump" shape feature). The total computation complexity is $kmT_{bs} + K(T_H + T_e) + T_{serial}$, where $K(T_H + T_e)$ contains the portion of computing for *Snapshot*. The speedup of distributed computing over single MATEX is,

$$Speedup = \frac{KmT_{bs} + K(T_H + T_e) + T_{serial}}{kmT_{bs} + K(T_H + T_e) + T_{serial}} \tag{11}$$

In R-MATEX, we have very small m. Besides, T_{bs} is larger than $T_H + T_e$. Therefore, the most dominating part is the KmT_{bs}. We can always decompose input transitions, and make k very small compared to K. Traditional method with fixed step size has N steps for the whole simulation. The complexity is $NT_{bs} + T_{serial}$. Then the speedup of distributed MATEX over the one with fixed step size is,

$$Speedup' = \frac{NT_{bs} + T_{serial}}{kmT_{bs} + K(T_H + T_e) + T_{serial}} \tag{12}$$

Usually, N is much larger than K and km. Uniform step sizes make N increased due to resolution of input transitions, to which K is not so sensitive. As mentioned before, k can be maintained in a small number. When elongating time span of simulation, N will increase. However, k will not change due to its irrelevant to time span (may bring more input transition features and increase computing nodes), and then $Speedup'$ tends to become larger. Therefore, our MATEX has more robust and promising theoretical speedups.

4. EXPERIMENTAL RESULTS

We implement our proposed MATEX in MATLAB R2013a and use UMFPACK package for LU factorization. The experiment is carried on Linux workstations with Intel Core$^{\text{TM}}$ i7-4770 3.40GHz processor and 32GB memory on each machine.

4.1 MATEX's Circuit Solver Performance

We test the part of circuit solver within MATEX using MEXP [15], as well as our proposed I-MATEX and R-MATEX. We create stiff RC mesh cases with different stiffness by changing the entries of matrix $\mathbf{C}$, $\mathbf{G}$. The stiffness here is defined as $\frac{Re(\lambda_{min})}{Re(\lambda_{max})}$, where λ_{min} and λ_{max} are the minimum and maximum eigenvalues of $-\mathbf{C}^{-1}\mathbf{G}$. Transient results are simulated in $[0, 0.3ns]$ with time step $5ps$. Table 1 shows the average Krylov subspace basis dimension m_a and the peak dimension m_p used to compute matrix exponential during the transient simulations. We also compare the runtime speedups (Spdp) over MEXP. Err (%) is the relative error opposed to BE method with a tiny step size $0.05ps$.

Table 1: Comparisons among MEXP, I-MATEX and R-MATEX with RC cases

Method	m_a	m_p	Err(%)	Spdp	Stiffness
MEXP	211.4	229	0.510	–	2.1×10^{16}
I-MATEX	5.7	14	0.004	2616X	
R-MATEX	6.9	12	0.004	2735X	
MEXP	154.2	224	0.004	–	2.1×10^{12}
I-MATEX	5.7	14	0.004	583X	
R-MATEX	6.9	12	0.004	611X	
MEXP	148.6	223	0.004	–	2.1×10^{8}
I-MATEX	5.7	14	0.004	229X	
R-MATEX	6.9	12	0.004	252X	

The huge speedups by I-MATEX and R-MATEX are due to large reductions of Krylov subspace basis m_a and m_p. As we known, MEXP is good at handling mild stiff circuits [15], but inefficient on highly stiff circuits. Besides, we also observe even the basis number is large, there is still possibility with relative larger error compared to I-MATEX and R-MATEX. The average dimension m_a of R-MATEX is a little bit larger than I-MATEX. However, the dimensions of R-MATEX used along time points spread more evenly than I-MATEX. In such small dimension Krylov subspace computation, the total simulation runtime tends to be dominated by matrix exponential evaluations on the time points with the peak basis dimension(m_p), which I-MATEX has more than R-MATEX. These result to a slightly better runtime performance of R-MATEX over I-MATEX. In large scale of linear circuit system and practical VLSI designs, the stiffness may be even more extensive and complicated. Many of them may also have large singular $\mathbf{C}$ and MEXP cannot handle without regularization process. These make I-MATEX and R-MATEX good candidates to deal with these scenarios.

4.2 Adaptive Time Stepping Comparisons

IBM power grid benchmarks [10] are used to investigate the performance of adaptive stepping TR (adpt) based on LTE controlling [9, 15] as well as the performance of I-MATEX and R-MATEX. Experiment is carried out on a single computing node. In Table 2, the speedups of I-MATEX is not as large as R-MATEX because I-MATEX with a large spectrum of $\mathbf{A}$ generates large dimension of Krylov subspace. In *ibmpg4t* case, I-MATEX and R-MATEX achieve maximum speedups resulted from relative small number points in GTS, which around 44 points, while the majority of others have over 140 points.

4.3 Distributed MATEX Performance

We focus on MATEX with R-MATEX in the following experiments with IBM power grid benchmarks. These cases have many input transitions (GTS) that limit step sizes of MATEX. Exploiting distributed computing, we decompose the input transitions, to obtain much fewer transitions of LTS for computing nodes. The input sources number is over ten thousand in the benchmarks, however, based on "bump" feature, we obtain a fairly small number of the required computing nodes, which is shown as *Group #* in Table. 3. To compete the baseline classical TR method with fixed time step $h = 10ps$, which requires 1000 pairs of forward and backward substitutions for the transient computing after factorizing $(\mathbf{C}/h + \mathbf{G}/2)$. In R-MATEX, $\gamma = 10^{-10}$ is set to sit among the order of varied time steps during the simulation. First, we pre-compute GTS and LTS groups and assign subtasks to corresponding nodes. MA-

Table 2: TR with adaptive stepping (TR(adpt)) vs. I-MATEX vs. R-MATEX. "Total(s)" is the total runtime of test cases. "DC(s)" records the time to obtain initial condition. Spdp[1]: Speedup of I-MATEX over TR(adpt); Spdp[2]: Speedup of R-MATEX over TR(adpt); Spdp[3]: Speedup of R-MATEX over I-MATEX.

Design	DC(s)	TR(adpt)	I-MATEX		R-MATEX		
		Total(s)	Total(s)	SPDP[1]	Total(s)	SPDP[2]	SPDP[3]
ibmpg1t	0.13	29.48	27.23	1.3X	4.93	6.0X	5.5X
ibmpg2t	0.80	179.74	124.44	1.4X	25.90	6.9X	4.8X
ibmpg3t	13.83	2792.96	1246.69	2.2X	244.56	11.4X	5.1X
ibmpg4t	16.69	1773.14	484.83	3.7X	140.42	12.6X	3.5X
ibmpg5t	8.16	2359.11	1821.60	1.3X	337.74	7.0X	5.4X
ibmpg6t	11.17	3184.59	2784.46	1.1X	482.42	6.6X	5.8X

Table 3: MATEX vs. TR ($h = 10ps$); Max. and Avg. Err.: maximum and average differences compared to all output nodes' solutions provided by IBM Power Grid Benchmarks; Spdp[4]: t_{1000}/t_{rmatex} transient stepping runtime speedups of MATEX over TR; Spdp[5]: t_{t_total}/t_{r_total} total simulation runtime speedups of MATEX over TR.

Design	TR		MATEX						
	$t_{1000}(s)$	$t_{t\_total}(s)$	Group #	$t_{rmatex}(s)$	$t_{r\_total}(s)$	Max. Err.	Avg. Err.	Spdp[4]	Spdp[5]
ibmpg1t	5.94	6.20	100	0.50	0.85	1.4E-4	2.5E-5	11.9X	7.3X
ibmpg2t	26.98	28.61	100	2.02	3.72	1.9E-4	4.3E-5	13.4X	7.7X
ibmpg3t	245.92	272.47	100	20.15	45.77	2.0E-4	3.7E-5	12.2X	6.0X
ibmpg4t	329.36	368.55	15	22.35	65.66	1.1E-4	3.9E-5	14.7X	5.6X
ibmpg5t	408.78	428.43	100	35.67	54.21	0.7E-4	1.1E-5	11.5X	7.9X
ibmpg6t	542.04	567.38	100	47.27	74.94	1.0E-4	3.4E-5	11.5X	7.6X

TEX scheduler is only responsible for simple superposition calculation at the end of simulation. Since MATEX slave nodes are in charge of all the computing procedures (Fig. 4) for transient simulation, and have no communications with each other during transient simulations, we can easily emulate such multiple-node environment using our workstations. We assign one MATLAB instance at each node of our workstations. After all MATEX slave nodes finish their jobs, we report the maximum runtime among these nodes as the total runtime t_{r_total} of MATEX. We also record "pure transient computing", the runtime of transient part t_{1000} and t_{rmatex} excluding LU, where t_{rmatex} is the maximum runtime of the counterparts among all MATEX nodes.

Our MATEX framework achieves 13X on average with respect to the pure transient computing t_{1000}/t_{rmatex} as well as 7X on the total runtime t_{t_total}/t_{r_total}. The average number of pairs of forward and backward substitutions for Krylov subspace generations is around 60 (km in Eq. (12)), while TR($h = 10ps$) has 1000 pairs (N in Eq. (12)) on each cases. The reductions of these substitutions bring large speedups in the pure transient computing. With huge reductions on these substitutions, the serial parts, including the operations for LU and DC, play more dominating roles in MATEX, which can be further improved by other more advanced methods.

5. CONCLUSIONS

We proposed a distributed framework MATEX for PDN transient simulation using the matrix exponential kernel. MATEX leverages the linear system's superposition property, and decomposes the task based on input sources features in order to reduce computational overheads for its subtasks at different nodes. We also address the stiffness problem for matrix exponential based circuit solver by rational Krylov subspace (R-MATEX), which has the best performance in this paper for adaptive time stepping without extra matrix factorizations. In IBM power grid benchmark, MATEX achieves 13X speedup over the fixed-step trapezoidal framework on average in transient computing after its matrix factorization. The overall speedup is around 7X.

6. ACKNOWLEDGE

We acknowledge the support from NSF-CCF 1017864. We also thank Ryan Coutts, Lu Zhang and Chia-Hung Liu for their helpful discussions.

7. REFERENCES

[1] M. Armbrust et al. A view of cloud computing. *CACM*, 53(4):50–58, 2010.

[2] M. A. Botchev. A short guide to exponential krylov subspace time integration for maxwell's equations. Dept. of Applied Mathematics, Univ. of Twente, 2012.

[3] Q. Chen, S.-H. Weng, and C. K. Cheng. A practical regularization technique for modified nodal analysis in large-scale time-domain circuit simulation. *IEEE TCAD*, 31(7):1031–1040, 2012.

[4] L. O. Chua and P.-M. Lin. *Computer Aided Analysis of Electric Circuits: Algorithms and Computational Techniques*. Prentice-Hall, 1975.

[5] T. A. Davis. *Direct Method for Sparse Linear Systems*. SIAM, 2006.

[6] B. Hindman et al. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, pages 22–22, 2011.

[7] P. Li. Parallel circuit simulation: A historical perspective and recent developments. *Foundations and Trends in Electronic Design Automation*, 5(4):211–318, 2012.

[8] 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.

[9] F. N. Najm. *Circuit simulation*. Wiley, 2010.

[10] S. R. Nassif. Power grid analysis benchmarks. In *ASPDAC*, pages 376–381, 2008.

[11] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 1992.

[12] Y. Saad. *Iteravite Methods for Sparse Linear Systems*. SIAM, 2003.

[13] J. van den Eshof and M. Hochbruck. Preconditioning lanczos approximations to the matrix exponential. *SIAM Journal on Scientific Computing*, 27(4):1438–1457, 2006.

[14] J. Wang and X. Xiong. Scalable power grid transient analysis via mor-assisted time-domain simulations. In *ICCAD*, pages 548–552, 2013.

[15] S.-H. Weng, Q. Chen, and C. K. Cheng. Time-domain analysis of large-scale circuits by matrix exponential method with adaptive control. *IEEE TCAD*, 31(8):1180–1193, 2012.

[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] J. Wilkinson. Kronecker's canonical form and the qz algorithm. *Linear Algebra and its Applications*, 28:285–303, 1979.

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

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

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

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

[22] H. Zhuang, S.-H. Weng, and C. K. Cheng. Power grid simulation using matrix exponential method with rational krylov subspaces. In *ASICON*, 2013.