Title	Circuit simulation via matrix exponential method for stiffness handling and parallel processing
Author(s)	Weng, SH; Chen, Q; Wong, N; Cheng, CK
Citation	The 30th IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2012), San Jose, CA., 5-8 November 2012. In ICCAD - IEEE / ACM International Conference on Computer-Aided Design Proceedings, 2012, p. 407-414
Issued Date	2012
URL	http://hdl.handle.net/10722/189851
Rights	ICCAD - IEEE / ACM International Conference on Computer-Aided Design. Proceedings. Copyright © IEEE Computer Society.

Circuit Simulation via Matrix Exponential Method for Stiffness Handling and Parallel Processing

Shih-Hung Weng[1], Quan Chen[2], Ngai Wong[2] and Chung-Kuan Cheng[1]
[1]Dept. of CSE, University of California San Diego, La Jolla, CA
[2]Dept. of EEE, University of Hong Kong, Hong Kong
email: s2weng@ucsd.edu, quanchen@eee.hku.hk, nwong@eee.hku.hk, ckcheng@ucsd.edu

***Abstract*— We propose an advanced matrix exponential method (MEXP) to handle the transient simulation of stiff circuits and enable parallel simulation. We analyze the rapid decaying of fast transition elements in Krylov subspace approximation of matrix exponential and leverage such *scaling effect* to leap larger steps in the later stage of time marching. Moreover, matrix-vector multiplication and restarting scheme in our method provide better scalability and parallelizability than implicit methods. The performance of ordinary MEXP can be improved up to 4.8 times for stiff cases, and the parallel implementation leads to another 11 times speedup. Our approach is demonstrated to be a viable tool for ultra-large circuit simulations (with 1.6M $\sim$ 12M nodes) that are not feasible with existing implicit methods.**

I. Introduction

Efficient yet accurate circuit simulation has always been one of the major demands in the IC design industry. The ever increasing size of circuitry in the advanced technology makes full-chip simulation a prohibitive task that requires days or even weeks to complete. Nowadays, the emerging multi-core system has opened new opportunities for researchers. Previous works [8], [9], [14], [21] have made efforts to speed up the circuit simulation by novel algorithmics or parallel techniques.

Circuit simulation involves solving a system of ordinary differential equations (ODEs), which are derived normally from Modified Nodal Analysis (MNA) and solved numerically in an explicit or implicit integration manner. The explicit methods require using unnecessarily small step sizes and are less attractive in general circuit simulation due to the stiffness of the ODE system, which results from a wide range of time constants of a circuit. On the other hand, the implicit methods, e.g., backward Euler and trapezoidal methods, can overcome the stiffness problem in ODE systems, and are widely adopted by conventional SPICE-like simulators. However, the implicit methods have to solve a linear system in each time step, which is usually by LU decomposition and requires complexity of O($n^{1.5}$) in both time and space, which cause the scalability problem for the implicit methods. Although iterative approaches can alleviate the memory requirement, solving an ill-conditioned matrix from the implicit methods is still a challenge.

Beyond the traditional explicit and implicit methods, a new class of explicit methods called exponential time differencing (ETD) has been drawing attention in the numerical community [2], [17]. The ETD methods analytically solve an ODE system within every discretized time step by directly computing the exponential of a matrix. In theory, the ETD methods avoid the local truncation error (LTE) of the polynomial expansion approximation, and the stability is the same as the trapezoidal method for passive systems. In application, the exponential of a matrix required in the ETD methods can be efficiently approximated by the Krylov subspace method, which involves mainly matrix-vector multiplication and has the advantages of scalability and parallelizability.

In [19], [20], Weng *et al.* have embraced the idea of ETD for the circuit simulation and proposed an adaptive step control scheme to enhance the performance. Although the ETD methods demonstrate their advantages, there are still two major limitations for ETD methods: 1) stiff circuits enforce the ETD methods to use small step sizes for reducing the approximation error of the Krylov subspace method, and thus the performance is damaged; 2) larger Krylov subspace bases required by stiff circuits, which usually need > 100 bases, pose a memory bottleneck for simulation involving millions of unknowns.

In this paper, we propose a *matrix exponential method* (MEXP) utilizing the *scaling effect* and restarting scheme to address these two limitations in stiff circuits. Specifically, our contributions are

- We reveal the scaling effect of the Krylov subspace method to enable the use of a larger step size when stepping forward.
- We utilize the restarting scheme to mitigate the memory usage when a large m is needed to strengthen the scaling effect.
- We demonstrate the parallelizability and scalability of MEXP by implementing on the GPU platform and testing with large-scale cases.

The experimental results show that the performance of MEXP for highly stiff circuits is improved up to 4.8 times by

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA

exploiting the scaling effect, and is accelerated in the GPU environment up to another 11 times. Furthermore, we demonstrate that MEXP is able to handle the circuit with up to 12 million nodes.

The rest of the paper is organized as follows. Sections II and III introduce the background of the matrix exponential method and our nonlinear circuit formulation, respectively. Section IV presents the adaptive and the restarting schemes in our matrix exponential method. Section V details the parallel implementation and Section VI shows experimental results. Finally, Section VII concludes the paper.

II. Preliminary

A. Matrix Exponential Method

In general, MNA represents a circuit by a system of ODEs as below:

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

where the matrix $\mathbf{C}$ describes the capacitances and inductances, the matrix $\mathbf{G}$ represents the resistances and the incidence between voltages and currents, and the matrix $\mathbf{B}$ indicates locations of the independent sources. The vector $\mathbf{x}(t)$ describes the nodal voltages and branch currents at time t, and the vector $\mathbf{u}(t)$ represents the input voltage and current sources. Given the initial value $\mathbf{x}(0)$ of the circuit, e.g., by DC analysis, the analytical solution [6] of (1) from t to $t+h$ is given by

$$\mathbf{x}(t+h) = e^{\mathbf{A}h}\mathbf{x}(t) + \int_0^h e^{\mathbf{A}(h-\tau)}\mathbf{b}(t+\tau)d\tau. \tag{2}$$

where $\mathbf{A} = \mathbf{C}^{-1}\mathbf{G}$ (we do not need to compute $\mathbf{C}^{-1}$ explicitly, as will be shown in Section II-C), and $\mathbf{b}(t) = \mathbf{C}^{-1}\mathbf{u}(t)$. Although $\mathbf{C}$ might be singular, we adopt a practical approach [5] that systematically regularizes $\mathbf{C}$ with affordable cost and acceptable sparsity.

Assuming that the input $\mathbf{u}(t)$ is piece-wise linear (PWL), the last term in (2) can be integrated analytically, turning the solution into the sum of three terms associated with matrix exponential operators:

$$\begin{aligned}\mathbf{x}(t+h) &= e^{\mathbf{A}h}\mathbf{x}(t) \\ &+ (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\mathbf{b}(t) \\ &+ (e^{\mathbf{A}h} - (\mathbf{A}h + \mathbf{I}))\mathbf{A}^{-2}\frac{\mathbf{b}(t+h) - \mathbf{b}(t)}{h}.\end{aligned} \tag{3}$$

We call the solution scheme based on this formulation as matrix exponential method, which is an *A-stable explicit* method because $\mathbf{x}$ approaches zeros as h tends to infinity when the real parts of $\mathbf{A}$'s eigenvalues are all negative.

Note that the simulation result of the matrix exponential method is exact when the matrix exponential is calculated exactly, and also the inputs satisfy the PWL assumption. In contrast, forward and backward Euler methods are the first order approximation of (2) while the trapezoidal method is accurate up to the second order.

B. One Matrix Exponential Formulation

Equation (3) and its extension of nonlinear formulation (10) have three matrix exponential terms, which are generally referred as φ-functions of the zero, first and second order [17]. It has been shown in [2, Theorem 2.1] that one can obtain the sum of them in one shot by computing the exponential of a slightly larger $(n+p)\times(n+p)$ matrix, where n is the dimension of $\mathbf{A}$ and p is the order of the φ-functions ($p=2$ in (3)). Thus, (3) can be rewritten into

$$\mathbf{x}(t+h) = \begin{bmatrix} \mathbf{I}_n & 0 \end{bmatrix} e^{\mathbf{A}'h} \begin{bmatrix} \mathbf{x}(t) \\ e_2 \end{bmatrix}, \tag{4}$$

with

$$\begin{aligned}&\mathbf{A}' = \begin{bmatrix} \mathbf{A} & \mathbf{W} \\ \mathbf{0} & \mathbf{J} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h} & \mathbf{b}(t) \end{bmatrix} \\ &\mathbf{J} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\end{aligned} \tag{5}$$

To keep the notations simple, we use $\mathbf{A}$ and $\mathbf{v}$ to represent $\mathbf{A}'$ and $[\mathbf{x}(t) \quad e_2]^{\mathsf{T}}$ in the following sections.

C. Krylov Subspace Approximation

The computation of the matrix exponential $e^{\mathbf{A}h}\mathbf{v}$ of (3) can be reduced using the Krylov subspace method [13], [18]. It projects $e^{\mathbf{A}}\mathbf{v}$ onto the Krylov subspace $\mathrm{K_m}(\mathbf{A}, \mathbf{v})$ with dimension of m and then only evaluates the exponential of an $m \times m$ Hessenberg matrix $\mathbf{H_m}$ constructed from Arnoldi process, which mainly involves matrix-vector multiplications and avoids computing $\mathbf{C}^{-1}$ explicitly by solving $\mathbf{C}^{-1}(\mathbf{G}\mathbf{v})$. Note that solving $\mathbf{C}$ is easier than $(\mathbf{C}/h + \mathbf{G})$ of implicit methods because $\mathbf{C}$ has fewer non-zeros and better structure that can favor direct and iterative sparse solvers.

The matrix exponential $e^{\mathbf{A}h}\mathbf{v}$ can be calculated as

$$\begin{aligned}e^{\mathbf{A}h}\mathbf{v} &\approx \mathbf{V_m}\mathbf{V_m}^{\mathsf{T}}e^{\mathbf{A}h}\mathbf{v} \\ &= \beta\mathbf{V_m}\mathbf{V_m}^{\mathsf{T}}e^{\mathbf{A}h}\mathbf{V_m}e_1 \\ &= \beta\mathbf{V_m}e^{\mathbf{H_m}h}e_1,\end{aligned} \tag{6}$$

where $\beta = \|\mathbf{v}\|_2$, $\mathbf{V_m}$ and $\mathbf{H_m}$ are obtained by Arnoldi process for $\mathbf{A}$ and $\mathbf{v}$, and e_1 is the first unit vector with dimension of $m \times 1$. Because m is usually small ($20 \sim 100$), the overall complexity of the exponential operator is greatly reduced. Hence, the scalability of MEXP is better than the traditional implicit methods. The error of the approximation (6) can be estimated by the following posteriori formula

$$err = \beta\eta|e_m^{\mathsf{T}}e^{\mathbf{H_m}h}e_1|, \tag{7}$$

where $\eta = \|\mathbf{H_m}(:, m)\|_2$ [13], [18].

D. Adaptive Step Size Control

MEXP can be further sped up by an adaptive step size strategy, which will enlarge (shrink) the time step size when the approximation error is less (larger) than a tolerance. Given the global error budget Tol and the total simulation time T, the

relation between the error and the tolerance can be represented as

$$err \leq h\frac{Tol}{T},$$

where err is calculated by (7). The scaling invariant property of the Krylov subspace method avoids the re-calculation of Arnoldi process and requires only evaluation of (6) with scaled $\mathbf{H_m}$. Step size h of MEXP can be fine tuned to satisfy the error tolerance without a significant cost. In contrast, in the implicit methods, the re-evaluation and error estimation for a new h have to again solve a linear system. Previous work [19], [20] showed that MEXP with adaptive step control leads to a several times speedup over the implicit methods for large-scale circuits.

E. Limitations

Although MEXP has advantages of accuracy, scalability and adaptivity, one major issue of the Krylov subspace method arises in highly stiff circuits, where the Krylov subspace method needs a large m or a small h to provide sufficient resolution to the spectrum of $\mathbf{A}h$. Besides that, the large storage of m bases for the high stiffness limits the application of large-scale circuit. Such issue of the Krylov subspace method restricts not only the performance but also the memory usage of MEXP for highly stiff circuits. We will address the issue in Section IV.

III. Nonlinear Circuit Formulation

For nonlinear circuits, the system of ODEs can be formulated as below:

$$\dot{\mathbf{q}}(\mathbf{x}(t)) + \mathbf{C}^l\dot{\mathbf{x}}(t) = (\mathbf{G}^l\mathbf{x}(t) + \mathbf{i}(\mathbf{x}(t))) + \mathbf{B}\mathbf{u}(t), \tag{8}$$

where $\mathbf{C}^l$ and $\mathbf{G}^l$ are the matrices representing linear components, and $\mathbf{q}$ and $\mathbf{i}$ are the charges and currents induced from nonlinear components. With a mild assumption that the charges in nonlinear elements behave linearly within the time step $[t, t+h]$, we can derive the above equation into the similar form as (1) by modeling $\dot{\mathbf{q}}$ as $\mathbf{C}^{nl}\dot{\mathbf{x}}$, where $\mathbf{C}^{nl}$ is the effective capacitance matrix for nonlinear elements within $[t, t+h]$. Then, the analytical solution of such nonlinear formulation can be written as

$$\begin{aligned}\mathbf{x}(t+h) &= e^{\mathbf{A}h}\mathbf{x}(t) \\ &+ \int_0^h e^{\mathbf{A}(h-\tau)}\left[\mathbf{F}(\mathbf{x}(t+\tau)) + \mathbf{b}(t+\tau)\right]d\tau,\end{aligned} \tag{9}$$

where $\mathbf{A} = \mathbf{C}_n^{-1}\mathbf{G}^l$, $\mathbf{C}_n = \mathbf{C}^l + \mathbf{C}^{nl}$, and $\mathbf{F}(\mathbf{x}(t)) = \mathbf{C}_n^{-1}\mathbf{i}(\mathbf{x}(t))$. By adapting the scheme of [16], we can decouple the nonlinear and linear terms. The second order implicit approximation is then of the form

$$\begin{aligned}\mathbf{x}(t+h) &= \frac{h}{2}\mathbf{F}(\mathbf{x}(t+h)) + e^{\mathbf{A}h}\left(\mathbf{x}(t) + \frac{h}{2}\mathbf{F}(\mathbf{x}(t))\right) \\ &+ (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\mathbf{b}(t) \\ &+ (e^{\mathbf{A}h} - \mathbf{A}h - \mathbf{I})\mathbf{A}^{-2}\Delta\mathbf{b}(t),\end{aligned} \tag{10}$$

where $\Delta\mathbf{b}(t) = \frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h}$. Since the nonlinear and linear terms are decoupled, we can compute those terms associated with the matrix exponential in advance, and then solve the nonlinear term by some iterative approaches, e.g., Newton's method or the fixed point method. The advantage behind this decoupling scheme is that iterations of solving $\mathbf{x}(t+h)$ involves no computation of the exponential of a matrix.

To provide higher capability for handling nonlinearity, we apply Newton's method to solve the nonlinear circuit, and the iteration equation for (10) is shown as follows:

$$\begin{aligned}&\left(\mathbf{C}_n + \frac{h}{2}\frac{\partial\mathbf{i}}{\partial\mathbf{x}}\bigg|_{\mathbf{x}_{n+1}^{k+1}}\right)\Delta\mathbf{x}_{n+1}^{k+1} = \\ &\left(\frac{h}{2}\mathbf{i}_{n+1}^k + \mathbf{C}_n\mathbf{x}_{n+1}^k - \mathbf{C}_n\mathbf{l}_{n+1}\right),\end{aligned} \tag{11}$$

where $\mathbf{l}_{n+1}$ represents the linear terms of (10), and $\frac{\partial\mathbf{i}}{\partial\mathbf{x}}$ is the Jacobian matrix. Such Jacobian matrix is equivalent to the effective conductance of nonlinear components at t_n, which is denoted as $\mathbf{G}^{nl}$. Therefore, instead of deriving the Jacobian matrix directly, as most SPICE-like simulators, we can construct $\mathbf{G}^{nl}$ via the inspection of linearized nonlinear components. Note that, in comparison with the implicit methods, e.g., backward Euler, the Jacobian matrix $(\mathbf{C}_n/h + \mathbf{G}^l + \mathbf{G}^{nl})$ will burden the linear system solver with much more non-zeros from $\mathbf{G}^l$.

IV. Scaling Effect and Restarted Matrix Exponential Method for Stiff Circuits

The scaling effect manifests the rapid decaying nature of fast transition components. Ideally, MEXP with the scaling effect can eventually step an arbitrarily large size even for stiff circuits. However, the interpolation error from the Krylov subspace method will prevent the optimal scaling effect and thus restrict the maximal step size.

In general, large m increases the number of interpolation points and reduces the interpolation error. We apply the restarting scheme to the Krylov subspace construction process so that the number of interpolation points increases effectively without adding the memory usage.

A. Scaling Effect in Matrix Exponential Method

With the one-exp formulation (4), time stepping in MEXP can be regarded as a series of product of matrix exponential, i.e., the solution at the $n+1$th step is related to the initial condition $\mathbf{x}_0$ by

$$\mathbf{x}_{n+1} = e^{\mathbf{A}h_{n+1}}e^{\mathbf{A}h_n}...e^{\mathbf{A}h_1}\mathbf{x}_0 \tag{12}$$

With the eigenvalue decomposition $\mathbf{A} = \mathbf{Q}_A\mathbf{\Sigma}_A\mathbf{Q}_A^{-1}$, (12) becomes

$$\mathbf{x}_{n+1} = \mathbf{Q}_A e^{\mathbf{\Sigma}_A h_{n+1}}e^{\mathbf{\Sigma}_A h_n}...e^{\mathbf{\Sigma}_A h_1}\mathbf{y}_0 = \mathbf{Q}_A e^{\mathbf{\Sigma}_A h_{n+1}}\mathbf{y}_n, \tag{13}$$

where $\mathbf{y}_i = \mathbf{Q}_A^{-1}\mathbf{x}_i, i = 0, 1, ...$ is the components of $\mathbf{x}_i$ on the eigenvectors space of $\mathbf{A}$ (which are referred as the eigencomponents of vector $\mathbf{x}_i$ hereafter).

Since a circuit intrinsically contains fast and slow transition elements, e.g. small and large capacitors, that correspond to different eigenvalues of $\mathbf{A}$, one can group the elements of $\mathbf{y}_n$, with a given threshold, and use $\mathbf{y}_n^f$ and $\mathbf{y}_n^s$ to represent the corresponding eigencomponents for the fast mode (negative eigenvalues with large magnitude) and the slow mode (negative eigenvalues with small magnitude), respectively. The effect of exponential shows the rapid decaying nature of fast transition elements (i.e., fast damping of negative eigenvalue with large magnitude in the exponent), and is reflected in the eigencomponents $\mathbf{y}_n^f$, which attenuate drastically as stepping forward.

In our MEXP, the Krylov subspace method still preserves such attenuation of the fast mode. The kth basis of the Krylov subspace at time T and in nth step can be represented as

$$\begin{aligned}(\mathbf{A}h_n)^k\mathbf{x}_n &= \mathbf{Q}_A(\mathbf{\Sigma}_A h_n)^k\mathbf{Q}_A^{-1}\mathbf{x}_n \\ &= \mathbf{Q}_A(\mathbf{\Sigma}_A h_n)^k\mathbf{y}_n \\ &= \mathbf{Q}_A(\mathbf{\Sigma}_A h_n)^k e^{\mathbf{\Sigma}_A T}\begin{bmatrix}\mathbf{y}_0^f \\ \mathbf{y}_0^s\end{bmatrix}. \end{aligned} \quad (14)$$

The fast mode eigencomponents $\mathbf{y}_0^f$ are attenuated rapidly by $e^{\mathbf{\Sigma}_A T}$. Although the power of $\mathbf{\Sigma}_A h_n$ acts as a counter force that brings those eigencomponents back to stage, the damping rate of exponential surpasses the increase by the power, so that the Krylov subspace method still benefits from the attenuation of $\mathbf{y}_0^f$ (as shown in Section VI-A). The rapid attenuation of eigencomponents implies that MEXP can alleviate the effect of stiffness, caused by the fast mode, and enables the use of larger h in the later stage of time marching where the step size should be more dominated by the slow mode of a circuit. We call this phenomenon as "scaling effect".

With the scaling effect, components of fast transition elements can be rapidly attenuated after a few steps. Smaller m, or larger h, can then be used in the Krylov subspace method at later steps. It should be noticed that for the nonlinear case, even though $\mathbf{A}$ is different every time step due to the nonlinear components, our formulation can still exploit the scaling effect. Since only linear terms are associated with the matrix exponential operator, without affecting Newton's iteration, we can calculate $\mathbf{l}_{n+1}$ separately in the form of (12) using the scaling effect.

Theoretically, according to (14), MEXP can use extremely large h in the later stage of simulation. Nevertheless, due to the approximation error of the Krylov subspace method, the allowable scaling of step size is restricted in practice. To provide an in-depth analysis of the error, we re-interpret the Krylov subspace approximation (6) from an interpolation perspective [18].

Lemma 4.1: The approximation (4) is mathematically equivalent to approximating $\exp(\mathbf{A})\mathbf{v}$ by $p_{m-1}(\mathbf{A})\mathbf{v}$, where p_{m-1} is the (unique) polynomial of degree $m-1$, which interpolates the exponential function in the Hermite sense on the set of Ritz values, the eigenvalues of $\mathbf{H}_m$, repeated according to their multiplicities.

The exact local error vector of approximation (4) can be written as

$$\mathbf{r} = e^{\mathbf{A}}\mathbf{v} - p_{m-1}(\mathbf{A})\mathbf{v} = \mathbf{Q}_A\left[e^{\mathbf{\Sigma}_A} - p_{m-1}(\mathbf{\Sigma}_A)\right]\mathbf{y} \quad (15)$$

Provided $\mathbf{A}$ is not highly nonnormal, i.e., the norm of $\mathbf{Q}_A$ remains reasonably bounded, the error mainly depends on $e^{\mathbf{\Sigma}_A} - p_{m-1}(\mathbf{\Sigma}_A)$, the mismatch between the exponential function and the interpolation function evaluated at the eigenvalues of $\mathbf{A}$, which we denote by the "interpolation error". The vector $\mathbf{y}$ denotes the eigencomponents of the starting vector $\mathbf{v}$.

B. Restarted Krylov Subspace Method

We adopt the restarted Krylov subspace method specific for matrix exponential computation [1], [10], [11]. Such restarting scheme mitigates the memory usage of the Krylov subspace method when a larger m is needed to strengthen the scaling effect. The Arnoldi process in computing matrix exponential is restarted every m iterations with the last basis vector from the previous cycle being the new starting vector.

$$\mathbf{v}_1^{(k)} = \mathbf{v}_{m+1}^{(k-1)} \quad (16)$$

The approximation $\mathbf{f}$ of the matrix exponential (multiplied with a vector) is updated with a correction term in each restarting.

$$\mathbf{f}^{(k)} = \mathbf{f}^{(k-1)} + \beta\mathbf{V}_m^{(k)}\left[e^{\hat{\mathbf{H}}_{km}}e_1\right]_{(k-1)m+1:km}, \quad (17)$$

where $\hat{\mathbf{H}}_{km}$ collects all the $\mathbf{H}_m$ from the k cycles of restarting

$$\hat{\mathbf{H}}_{km} = \begin{bmatrix} \mathbf{H}_m^{(1)} & & & \\ \mathbf{E}_m^{(2)} & \mathbf{H}_m^{(2)} & & \\ & \ddots & \ddots & \\ & & \mathbf{E}_m^{(k)} & \mathbf{H}_m^{(k)} \end{bmatrix}, \quad (18)$$

where $\mathbf{E}_m^{(k)} = \eta_k e_m^T e_1$. The posterior error of (17) can be estimated by

$$e^{\mathbf{A}}\mathbf{v} - \mathbf{f}^{(k)} = \eta_k \sum_{j=1}^{2}\left[e_{km}^T\phi_j\left(\hat{\mathbf{H}}_{km}\right)e_1\right]\omega_{j-1}(\mathbf{A})\mathbf{v}_{m+1}^{(k)}, \quad (19)$$

where ϕ_j is the divided differences of the exponential function and ω_j is the nodal function w.r.t. the minimal and maximal eigenvalues of $\mathbf{H}_m^{(k)}$.

It is shown in [11] that the k cycles of this restarting is equivalent to interpolating the exponential function (in the Hermite sense) at the *union* of the k set of eigenvalues of $\mathbf{H}_m$. This way, a much larger "effective" m is allowed without increasing the memory demand. In principle, given a sufficient number of restarting, the restarted method is able to work with arbitrary step size.

One major shortcoming of the above restarting scheme lies in calculation of the exponential of a matrix whose complexity increases with km. As a consequence, we would like to limit the number of restarting k at each step and adjust the step size adaptively to meet the accuracy requirement. This naturally calls for an integration of the restarted Krylov

subspace method and the adaptive step control based on the scaling effect. Such integration utilizes the restarting scheme to reduce the interpolation error so that the scaling effect can be manifested and exploited by adaptively stepping.

C. Overall Restarted Algorithm

Algorithm 1: Restarted MEXP

Input: $\mathbf{C}$, $\mathbf{G}$, $\mathbf{B}$, $\mathbf{u}(t)$, initial h, initial m, total error budget TOL and total time t_f
Output: $\mathbf{x}(t)$
$t = 0$; $\mathbf{x}(0)$ = DC_analysis;
while $t \leq T$ **do**
  [$\mathbf{C}_r$, $\mathbf{G}_r$, $\mathbf{B}_r$] = regularization($\mathbf{C}$, $\mathbf{G}$, $\mathbf{B}$);
  Compute $\mathbf{x}_{new}$ by restarted Krylov subspace method (17);
  Estimate err by (19);
  $tol = \frac{h}{t_f} TOL$;
  if $err > tol$ **then**
    Reduce h and re-compute $\mathbf{x}_{new}$;
  else if $err < tol$ **then**
    Estimate h_{new} by repeated tuning to fully use error margin;
  else
    h_{new} is unchanged
  end
  $t = t + h$;
  $\mathbf{x}(t) = \mathbf{x}_{new}$;
  $h = h_{new}$;
end

The context of restarted MEXP follows the MEXP described in Section II. The step size is adaptively adjusted according to the posterior error estimate. The two main distinctions lie in that: 1) the maximal number of restarting k_{max} is fixed to a small value, e.g., 5, in each step. It is intended to enlarge the effective m to roughly $k_{max}m$, without inducing too much overhead from the evaluation of $e^{\hat{\mathbf{H}}_{km}}$; 2) unlike in the unrestarted case, there is no fast re-evaluation once h is changed, since the basis vectors $\mathbf{V}_m$ from previous restarting cycles, which are not stored to save memory, are all required to generate an updated approximation $\mathbf{f}$. Hence, we only employ re-evaluation(s) when the error exceeds the tolerance. If the step size can be increased owing to a small error, we use the step size in the next step, instead of applying it right in the current step with re-evaluation. In this way, we could still benefit from the scaling effect while minimizing the number of re-evaluations. The posterior error estimate with a changed h is calculated by (19), with the storage of some auxiliary quantities. The whole restarted MEXP is summarized in Algorithm 1.

V. Parallel Restarted Matrix Exponential Method

In this section, we present the parallel version of MEXP. We focus on the sparse matrix-vector multiplication—one of the key components in restarted MEXP that shows strong potential for parallelism. The basic idea of parallel sparse matrix-vector multiplication is to simultaneously calculate each row of the product vector. Many researchers [3], [4], [15] have investigated and parallelized such matrix arithmetic on different environments, such as FPGA, cluster and GPU. In this paper, we target the parallel restarted MEXP using the GPU platform for two reasons. First, GPU has better cost-toperformance ratio. Designers can adopt the parallel restarted MEXP with affordable cost. Second, the communication overhead of the sparse matrix-vector multiplication is mitigated since the communication is now inter-thread instead of intermachine.

Our parallel restarted MEXP uses a hybrid CPU-GPU implementation. We only parallelize Arnoldi process and matrix exponential operation of a smaller matrix while keeping other operations serial on CPU. For Arnoldi process, the parallel matrix-vector multiplication is implemented by [3]. Although the parallel sparse matrix-vector multiplication has up to an order of magnitude speedup comparing to CPU implementation, the limited memory on GPU ($2GB \sim 4GB$) imposes a restriction on the dimension of Krylov subspace method for large-scale circuits. Fortunately, with the restarting scheme, MEXP method can make the effective m sufficiently large under restricted memory resource.

In the computation of matrix exponential, even though the reduced matrix by Krylov subspace method can be efficiently evaluated on CPU, the restarting scheme would increase the dimension of matrix up to hundreds, and the performance of evaluation on CPU will significantly drop. We implement the parallel matrix exponential based on a scaling and squaring method [12], which involves only basic dense matrix arithmetic that has already been optimized in the GPU environment [15].

Besides the parallelization on GPU, we minimize the data transfer cost between GPU and CPU that is one potential performance hazard of the hybrid implementation. To minimize the memory transfer, we keep the intermediate matrices, e.g., $\mathbf{V}_m$ and $\mathbf{H}_m$, on GPU, and consecutively execute Arnoldi process and matrix exponential computation. Thus, we only transfer the solution vector of the next time step back to CPU. For linear circuits, we transfer $\mathbf{G}$ and $\mathbf{L}$ and $\mathbf{U}$ decomposed from $\mathbf{C}$ at the beginning of MEXP since those linear elements remain the same during simulation. For nonlinear circuits, even though $\mathbf{C}$ of every time step changes, we do not have to transfer matrices for every Newton's iteration. This is because we decouple the linear and nonlinear terms, and thus, at each time step, only one data transmission for $\mathbf{C}$ is required. The execution flow of Algorithm 1 between CPU and GPU are shown in Figure 1.

We would like to mention that the backward and forward substitutions for $\mathbf{L}$ and $\mathbf{U}$ are also parallelized on GPU [15]. For large-scale circuit that cannot be decomposed, we adopt iterative approaches that are also based on matrix-vector product, and then solve $\mathbf{C}$ on GPU in parallel.

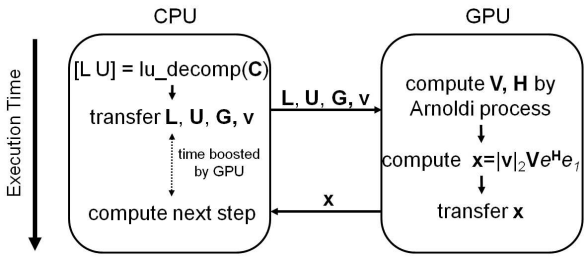

Fig. 1. Execution flow between CPU and GPU.

VI. Numerical Results

The experiments are performed on a Linux machine with an Intel i7 2.67GHz processor and 4GB memory. The parallel operations are implemented on a NVIDIA C1060 device with Tesla T10 architecture and maximum 77.6 GFLOPs for double precision. The MEXP method is prototyped in MATLAB, and the parallel restarting Krylov subspace method and matrix exponential computation are implemented in CUDA. The LU decomposition is performed by KLU package [7].

A. A Case Study of Scaling Effect

We analyze the scaling effect with the following simple model problem mainly for the ease of reproduction. One can also use a RC ladder and obtain similar observations. Let $\mathbf{A}$ be a 100×100 diagonal matrix with the diagonal elements range logarithmically in $-\left[10^{-2}, 10^{2}\right]$, i.e. $-logspace(-2, 2, 100)$, and $\mathbf{x}_0 = [1, 1, ..., 1]^T / \sqrt{100}$. We run 100 steps in the form of (12), and record for each step the eigencomponents of the solution vector ($\mathbf{y}$) and $\mathbf{r}$ in (15). Adaptive step is applied based on the posterior error estimate (constant local tolerance is used $tol = 10^{-6}$). The number of interpolation points $m = 10$. Fig. 2 shows the eigencomponents for 6 selected eigenvalues with different magnitudes.

In the upper subfigure of Fig. 2, the eigencomponents for fast mode (negative eigenvalues with large magnitude) generally attenuate rapidly, while the eigencomponents for slow mode (the two smallest eigenvalues) stay nearly constant. The relatively slow and oscillating drop of the eigencomponent of -100 is due to the oscillation of the maximal Ritz value [11], which induces large interpolation error and weaker attenuation when the two values are out of phase. In the lower subfigure of Fig. 2, the eigencomponents of large eigenvalues in the error vector are heavily attenuated, while the contribution from small eigenvalues remains at the same order, indicating that the error in later stage is largely determined by the small eigenvalues. The attenuation of some components in the error vector reduces the norm of error and thus allows the usage of gradually increasing h in later steps under the same tolerance, which is shown in Fig. 3.

In Fig. 3, when m is set as 10, the sum of h over 100 steps is 68.42, and the ratio of the initial h to the largest allowable h is 22.55. When we increase m to 15, the increased m reduces the interpolation error and thus allows a larger step size in each step. The sum of h for $m = 15$ achieves 196.52, which

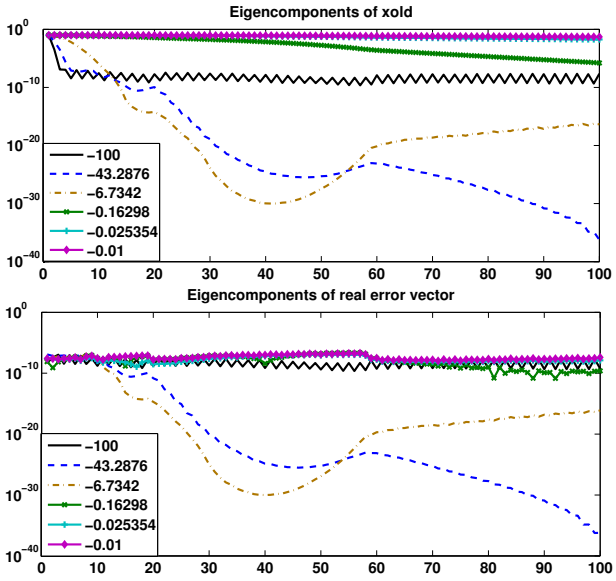

Fig. 2. Eigencomponent vs. # of steps ($m = 10$, fixed $tol = 10^{-6}$)

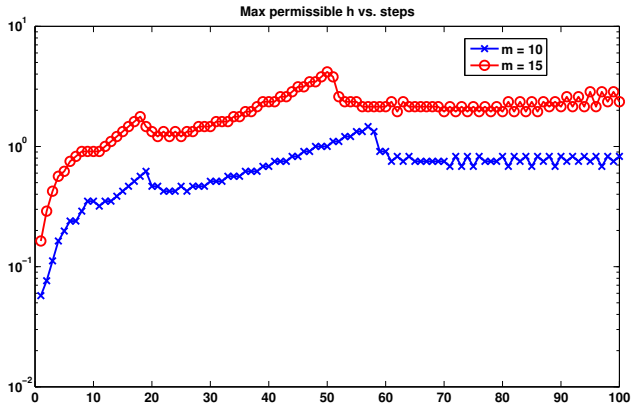

Fig. 3. Max permissible h vs. steps for $m = 10$ and $m = 15$ (fixed $tol = 10^{-6}$)

is nearly triple of that for $m = 10$, with just a half increase of computational cost, while the ratio of the initial and the largest allowable h is still 22.55. Therefore, it is beneficial to use a larger m in large, stiff, problems, realized by the restarting of Krylov subspace method.

B. Performance of Restarted MEXP

Table I details the functionality, size, type (L for linear and NL for nonlinear), stiffness of cicuits and also the number of nodes without grounded capacitance for each benchmark circuit. We represent the stiffness of a circuit with the largest generalized eigenvalue of the matrix pencils ($\mathbf{G}$, $\mathbf{C}$). Highly stiff circuits have a value ranged from $10^{16} \sim 10^{20}$. Table II records the result of regularization [5] for the cases (D2, D3, and D6). From the table, we can see that the number of nonzeros of $\mathbf{C} + \mathbf{G}$ before and after the regularization

process is not affected signifiantly and even decreased because some elements are eliminated. Furthermore, the regularization process still maintains a reasonable condition number of $\mathbf{C}_r$ for inverse and spends acceptable runtime for each cases.

TABLE I

SPECIFICATIONS OF BENCHMARK CIRCUITS

Design	Category	Type	Nodes	Nodes w/o Cap.	Stiffness
D1	power grid	L	2.5K	0	3.9×10^{17}
D2	trans. line	L	5.6K	431	1.6×10^{19}
D3	ALU	NL	10K	373	8.7×10^{18}
D4	IO	NL	630K	0	1.6×10^{20}
D5	power grid	L	800K	0	2.6×10^{14}
D6	power grid	L	1.6M	0.6M	1.6×10^{17}
D7	power grid	L	12M	0	2.6×10^{14}

TABLE II

RESULT OF REGULARIZATION

Design	cond($\mathbf{C}_r$)	nnz($\mathbf{C}+\mathbf{G}$)	nnz($\mathbf{C}_r+\mathbf{G}_r$)	runtime
D2	1.1×10^7	0.9M	0.9M	5.9s
D3	4.4×10^5	44K	43K	1.2s
D6	1.4×10^6	5.4M	4.8M	191.2s

Table III shows the performance gained from the scaling effect. We compare the performance of ordinary MEXP with adaptive step size (MEXP) and restarted MEXP with both restarting and adaptive control (RMEXP) as outlined in Algorithm 1, where the number of restarting is 5. In addition, we implement the trapezoidal method (TRAP) with adaptive step control as the baseline performance of the circuit simulation. All three methods adopt the same adaptive scheme in Algorithm 1, and the total error budget TOL is 10^{-4} for all cases. The total simulation time and the initial step size for each case are denoted in columns t_f and "init. h", respectively.

For MEXP and TRAP, TRAP outperforms MEXP only in small case D1 because a linear system can be solved efficiently in such scale. As the size of circuit becomes large, TRAP is slowed down by solving a large linear system while MEXP benefits from the scaling effect and the sparse matrix-vector multiplication. For stiff cases D2, D3 and D4, MEXP achieves a $2.06\times$ speedup over TRAP by the scaling effect on average. For moderately stiff cases (D5 and D7), since MEXP can use much larger step size than that in stiff cases, MEXP outperforms TRAP by over 100 times. MEXP also demonstrates the scalability in the cases with millions of nodes (D6 and D7), while TRAP encounters the scalability issues in runtime and memory. Notice that although both MEXP and TRAP have to perform Newton's method for nonlinear circuits, the Jacobian in our formulation has much fewer non-zeros than that in the traditional implicit methods. For example, in D4, solving the Jacobian in MEXP takes only 0.73 seconds whereas TRAP requires 6.86 seconds due to those extra non-zeros.

Our restarting scheme can further improve the scaling effect in RMEXP. For linear cases with high stiffness (D1, D2 and D6), RMEXP improves the performance over MEXP by up to $4.1\times$. For the nonlinear cases (D3 and D4), our nonlinear formulation in RMEXP still takes advantage of the scaling effect that improves the performance up to 4.8 times, because the decoupling scheme minimizes the calculation of the matrix exponential during Newton's iterations. Overall, RMEXP achieves $3.6\times$ speedup over MEXP on average, and achieves an average of $8.25\times$ speedup over TRAP on stiff cases D2, D3 and D4.

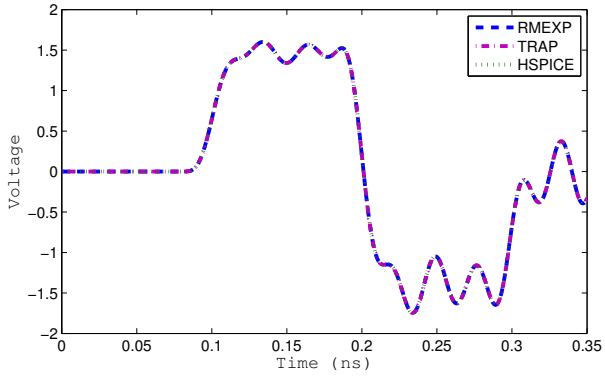

Fig. 4. Results of D5 by RMEXP and HSPICE

Fig. 4 shows the waveforms by TRAP and RMEXP as well as the golden reference waveform by HSPICE for testcase D5. The errors to the golden reference for both waveforms of TRAP and RMEXP are 5.78×10^{-4} and 2.02×10^{-4}, respectively.

Fig. 5 shows the speedup of the parallel Krylov subspace method (Krylov-GPU) and computation of $e^{\mathbf{H_m}}$ (exp($\mathbf{H_m}$)) over the serial version. For small matrices, Krylov-GPU and exp($\mathbf{H_m}$) accelerate little in the throughput-oriented GPU. For example, Krylov-GPU and exp($\mathbf{H_m}$) gain $0.9\times$ and $0.5\times$ speedup for the matrix size of 10K and 20. As the matrix size goes up, the massive parallelism of GPU surpasses faster floating point operations of CPU. Then, both operations can achieve $12\times$ and $10\times$ speedup for the size of 10M and 2000. Overall, with integrating both parallel operations into Algorithm 1, the performance of parallel restarted MEXP (RMEXP-GPU) (also shown in Table III) presents an average

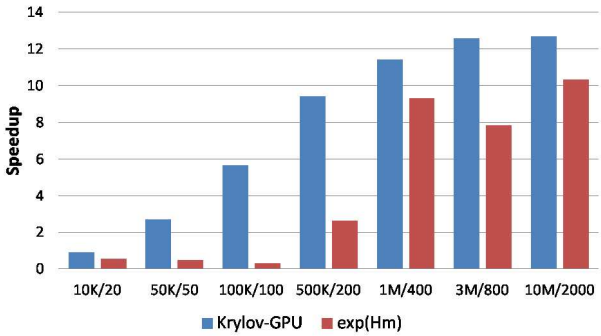

Fig. 5. Speedup of core operations by GPU

TABLE III

PERFORMANCE COMPARISON (SAME ADAPTIVE STEP SCHEME WITH TOL = 10^{-4} FOR ALL METHODS AND CIRCUITS)

Design	t_f	init. h	m	TRAP	# steps	MEXP	# steps	RMEXP	# steps	RMEXP-GPU	comm.
D1	1ns	1ps	30	67.85s	151	475.46s	2,595	199.10s	143	348.55s	0.8ms
D2	100ps	0.1ps	30	3,085.91s	390	2,113.51s	165	891.03s	56	982.14s	41.1ms
D3	100ps	0.1ps	30	8,053.45s	451	2,502.30s	285	572.54s	43	535.93s	53.7ms
D4	10ps	0.01ps	20	10,071.40s	357	6,646.38s	333	1,384.35s	92	194.11s	396.0ms
D5	1ns	1ps	20	16,431.12s	483	159.73s	249	83.20s	47	7.20s	121.5ms
D6	10ps	0.01ps	30	>1 day	n/a	12,626.88s	133	3,114.84s	34	239.46s	189.8ms
D7	1ns	1ps	20	fails	n/a	12,498.83s	223	7,821.65s	74	629.56s	1478.6ms

of $11\times$ speedup for large-scale cases, compared with the serial RMEXP.

Take the largest case D7 as an example. TRAP fails the simulation due to insufficient memory while MEXP and RMEXP require about 3.5 and 2.2 hours, respectively. RMEXP-GPU shows the simulation takes only 10 minutes using GPU. The speedup by GPU over MEXP and RMEXP are $19.9\times$ and $12.4\times$, respectively.

Note that the communication overhead between CPU and GPU is negligible since we minimize the data transmission in the hybrid CPU-GPU architecture. Although, for nonlinear circuits, the change of $\mathbf{C}$ and $\mathbf{G}$ by nonlinear devices at every time step increases the communication between CPU and GPU, such overhead is still negligible. The column "comm." in Table III shows the time of transferring $\mathbf{L}$, $\mathbf{U}$ and $\mathbf{G}$ once from CPU to GPU. We can see that even the largest case D7 requires only about 1.5 seconds. This is because the sparsity of the matrices in the circuit simulation application greatly reduces the transferred data, and also KLU [7] maintains acceptable sparsity in both $\mathbf{L}$ and $\mathbf{U}$ matrices.

VII. CONCLUSION

In this paper, we propose a parallel and restarted matrix exponential to utilize the scaling effect, which can overcome the stiffness of circuitry. The scaling effect enables the use of gradually larger step size as time frame marches forward. Moreover, our method has better scalability and parallelizability, which are the major limitations of existing implicit methods. The experimental results show that MEXP has a speedup over the trapezoidal method in orders of magnitude for cases with millions of nodes. For stiff circuits, our restarted MEXP improves the performance of MEXP by up to 4.8 times, and the performance can be further accelerated by up to another 11 times by GPU parallelization. For cases with millions of nodes, our method is able to simulate with acceptable runtime and memory usage while the trapezoidal method breaks down. The superior scalability and parallelizability of the proposed method enable a substantial expansion of computational capability for modern extremely large circuit simulation problems.

VIII. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of NSF CCF-1017864, Hong Kong General Research Fund (GRF) project 718711E and Hong Kong University Grant Council (AoE/P-04/08).

REFERENCES

[1] M. Afanasjew, M. Eiermann, O. G. Ernst, and S. Guttel. Implementation of a restarted Krylov subspace method for the evaluation of matrix functions. *Linear Algebra and its Applications*, 429(10):2293 – 2314, 2008.

[2] A. H. Al-Mohy and N. J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM Journal on Scientific Computing*, 33(2):488–511, 2011.

[3] M. Baskaran and R. Bordawekar. Optimizing sparse matrix-vector multiplication on gpus. *IBM research report RC24704, IBM*, 2009.

[4] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *SC*, page 18. ACM, 2009.

[5] 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 Trans. on Computer-Aided Design*, 31(7):1031–1040, 2012.

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

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

[8] A. Devgan and R. A. Rohrer. Event driven adaptively controlled explicit simulation of integrated circuits. In *ICCAD*, pages 136–140, 1993.

[9] W. Dong and P. Li. Parallelizable stable explicit numerical integration for efficient circuit simulation. In *DAC*, 2009.

[10] M. Eiermann and O. G. Ernst. A restarted Krylov subspace method for the evaluation of matrix functions. *SIAM J. NUMER. ANAL*, 44:2481–2504, 2006.

[11] M. Eiermann, O. G. Ernst, and S. Guttel. Deflated restarting for matrix functions. *SIAM J. MATRIX ANAL. APPL.*, 32(2):621–641, 2011.

[12] N. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1196, 2005.

[13] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 34(5):1911–1925, 1997.

[14] P. Li. Parallel circuit simulation: A historical perspective and recent developments. *Foundations and Trends in Electronic Design Automation*, 5(4):211–318, 2001.

[15] R. Nath, S. Tomov, and J. Dongarra. Accelerating gpu kernels for dense linear algebra. *High Performance Computing for Computational Science–VECPAR 2010*, pages 83–92, 2011.

[16] Q. Nie, Y. Zhang, and R. Zhao. Efficient semi-implicit schemes for stiff systems. *Journal of Computational Physics*, 214(2):521–537, 2006.

[17] J. Niesen and W. M. Wright. A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators. *ACM Trans. Math. Software*. in press.

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

[19] S.-H. Weng, Q. Chen, and C.-K. Cheng. Circuit simulation using matrix exponential method. In *ASICON*, pages 369–372. IEEE, 2011.

[20] S.-H. Weng, Q. Chen, and C.-K. Cheng. Time-domain analysis of large-scale circuits by matrix exponential method with adaptive control. to be appeared in *IEEE Trans. on CAD*, 2011.

[21] X. Ye, W. Dong, P. Li, and S. Nassif. Maps: Multi-algorithm parallel circuit simulation. In *ICCAD*, pages 73–78, 2008.