Fast Current Constraints Generation for Chip Safety

Cedric Feghali
ECE Dept, University of Toronto
Toronto, Ontario, Canada
cedric.feghali@mail.utoronto.ca

Farid N. Najm
ECE Dept, University of Toronto
Toronto, Ontario, Canada
f.najm@utoronto.ca

***Abstract*—Electromigration is a reliability concern that affects chip power grids carrying high current over the course of several years. In order to guarantee safety of the chip from void nucleations due to electromigration, a set of current constraints can be generated as a guideline for chip design. In this work, we propose a tool to efficiently generate approximate constraints through model order reduction, based on Arnoldi's algorithm. Our method leverages several properties of the matrices at hand to efficiently perform the required computations in the reduced subspace, before projecting the results back to the original space.**

***Index Terms*—Electromigration, power grid, void, current constraints, model order reduction, Arnoldi.**

I. Introduction

Electromigration (EM) is the movement of metal atoms in the direction of the electron wind in metal lines carrying high current. The problem is exacerbated whenever the current density is high in the line, which is getting more common with continued technology scaling. It is also affected by different parameters such as temperature, the materials used, etc. EM can ultimately lead to the formation of *voids* and *hillocks* in the metal line. A void is the absence of material at some point in the line, whereas a hillock is the extrusion of metal through cracks in the dielectric. Hillocks are not really a concern in modern damascene metal systems due to the metal liner that blocks these extrusions, which is why we focus only on voids. While atoms are moving, and before the void appears, a stress gradient develops in the line, with tensile stress at one end of the metal line and compressive stress at the other end. This is called the *void nucleation phase*. Whenever the tensile stress exceeds a critical stress threshold called σ_{crit}, a void "nucleates" and appears at the junction, marking the start of the *void growth phase*. The presence of a void may completely block atomic flow through that line, but still allows electric flow through the less conductive metal liner. During void growth, the resistance of the line increases with void length until it reaches some steady-state value. With voids nucleating in different metal lines, the resulting resistance changes can cause circuit failures.

Although EM is common in many other structures, the power delivery network (PDN), and more specifically the on-die power grid, will be the focus of our work, because it mostly carries unidirectional current. Because electromigration is a very slow phenomenon which takes months and even years to develop, high-frequency components of the current are not relevant. We instead consider the DC average of the currents in the power grid, which are readily available through the power budgets of the underlying circuit blocks drawing power from the grid.

This work was supported in part by the Semiconductor Research Corporation (SRC) and by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

The power grid is composed of metal meshes on different metal layers connected to each other by means of vias. Within each layer, the metal in the supply network consists of physically disconnected metal structures that are generally shaped like trees (i.e., graphs with no cycles), which are called *interconnect trees* in the EM literature [1]. Even though cycles might be present, it is common practice in the field to assume that the grid consists only of trees. Every tree consists of a number of connected straight-line metal *branches*. The end-points of branches are called *junctions*, which may be shared among multiple connected branches. Since the grid is formed of trees, each tree with $m \geq 2$ junctions has $(m-1) \geq 1$ branches. Every junction is connected to at least one metal branch, and most junctions are connected to vias which in turn connect them to other metal layers. Since vias do not allow the flow of metal atoms, electromigration analysis may consider a via to be connected to a current source rather than another line in a different layer. Moreover, the power grid must be able to drive a load, which is modeled as ideal current sources that do not vary with time, and that are connected to the lowest layer of the grid through junctions.

With voids nucleating in different areas of the power grid, and with the resulting resistance changes, some junction voltages will drop, which may lead to voltage drop violations and timing failures. Power grid safety under electromigration is a major reliability concern for chip design, which can be addressed in multiple ways. This work focuses on what has been called the "inverse problem" [2]. Given a grid, we wish to find constraints for the current sources that load the grid, which would guarantee safety of the chip from EM for the desired lifetime, and would serve as a guideline for chip design. Based on a closed-form expression for the currents constraints derived in [2] under certain assumptions, we introduce a fast and efficient way to compute the constraints using a model order reduction technique. This allows us to handle much larger interconnect trees compared to existing methods.

II. Background

A. Equivalent Circuit Model

In [3], a model was developed to track the evolution of stress in a power grid by drawing an analogy with specially constructed RC circuits. The equivalent circuit is constructed as follows: every tree junction corresponds to a node in the circuit, and every tree branch corresponds to a series chain of resistors. Each chain is the result of the discretization of an RC transmission line, with a discretization size of 20. We therefore make a distinction between two different types of nodes: the *junction nodes*, which correspond to the original junctions of the tree, and the *internal nodes* which are a result of the discretization of a branch. In the new RC circuit, there are $n = 20(m-1)+1$ nodes, m of which are junction nodes, and the rest are internal nodes. As previously mentioned, we represent the via connections by ideal current sources, so in the equivalent circuit, every junction node is also connected to an ideal *current source* to ground, whose values are stored in the $m \times 1$ vector u. The entries of u are defined as a function of the current source vector i of the original tree,

$$u = MDM^{+}i. \tag{1}$$

In this equation, the $m \times (m-1)$ matrix M is the incidence matrix of the tree, whose (i,j)th entry is 0 if node i is not an endpoint of edge j, and otherwise $+1$ (or -1) if the direction of the edge is *away from* (or *towards*) node i, and M^{+} refers to the Moore-Penrose inverse of M. As for D, it is the diagonal matrix of the strictly positive scale factors used in [3, eq. 35] to multiply the branch currents of the original tree.

The key finding in [3] is that the stresses in the original interconnect tree can be found by simply scaling the voltages in the equivalent circuit. In other words, one can use traditional circuit techniques to solve the circuit equations and obtain the node voltages $v(t)$, then deduce the values of the stresses $\sigma(t)$ in the original tree according to

$$v(t) = \xi\sigma(t), \tag{2}$$

where $\xi = 1$ V/MPa, as in [3, eq. 31].

Let us now define the $n \times n$ conductance matrix G of the equivalent circuit, the $n \times n$ diagonal matrix C of capacitance values, and the $n \times m$ matrix H such that its (i,j)th entry is 1 if node i is connected to current source j (and 0 otherwise). After applying nodal analysis on the equivalent circuit, we can construct the stress equation

$$G\sigma(t) + C\dot{\sigma}(t) = Hu = HMDM^{+}i, \tag{3}$$

or

$$\dot{\sigma}(t) = A\sigma(t) + C^{-1}HMDM^{+}i, \tag{4}$$

where $A = -C^{-1}G$ is the *system matrix*.

B. Safety from Voids

Even though the time-to-failure (TTF) of a metal line or metal network is stochastic due to random process variations, we focus only on specific (deterministic) grid instances. Given a specific instance of a power grid, we call the *set of safe current assignments* all those underlying currents for which the lifetime of the chip is greater than or equal to the desired target lifetime T, i.e.,

$$\mathbb{S}(T) = \{i \in \mathbb{R}^m : \text{TTF}(i) \geq T\}. \tag{5}$$

For the sake of generating current constraints, we make a simplifying assumption by considering safety of the chip as *safety from voids*, instead of safety from voltage drop violations. This allows us to consider trees independently, since no structural change occurs before void nucleation. If all trees in the grid are safe, we can declare that the grid is safe as well. We rely on the fact that if the stress does not exceed the stress threshold anywhere, no voids will appear. Since this condition is hard to guarantee at every time point from time 0 until the target lifetime T, we make another simplifying assumption: we assume that the stresses are monotone. Although this does not necessarily always hold, it is almost always the case except for initial dynamics, or after a void formation that induces structural change (which would not happen before failure when considering safety from voids).

Under these assumptions, the safety set is expressed as [2]

$$S(T) = \{i \in \mathbb{R}^m : \sigma(T) \leq \sigma_{\text{crit}}\}. \tag{6}$$

As shown in [2], the stress at time T can be expressed in terms of the original input currents i and the initial stress vector σ_0 as

$$\sigma(T) = e^{AT}\sigma_0 + (I - e^{AT})G^{+}HMDM^{+}i, \tag{7}$$

so that

$$S(T) = \{i \in \mathbb{R}^m : Ki \leq s\}, \tag{8}$$

where the pair (K, s) are the $n \times m$ matrix K and the $n \times 1$ vector s, defined as

$$K = (I - e^{AT})G^{+}HMDM^{+} \quad \text{and} \quad s = \sigma_{\text{crit}} - e^{AT}\sigma_0. \tag{9}$$

The most significant challenge in the generation of the constraints is the computation of the matrix exponential e^{AT}, as will be explained in Section II-C.

C. The Matrix Exponential

Let A be a square $n \times n$ matrix, and T be a time scalar which multiplies the matrix A. The exponential of AT, written e^{AT}, is the $n \times n$ matrix defined as

$$e^{AT} \triangleq I + AT + \frac{A^2T^2}{2!} + \frac{A^3T^3}{3!} + ... = \sum_{k=0}^{\infty} \frac{A^kT^k}{k!}. \tag{10}$$

An algorithm to compute the matrix exponential can be developed based on this definition, but it is known that the matrix computed by this algorithm, which is meant to approach e^A, is unreliable and takes a very long time to converge. There is no guaranteed error bound, and the algorithm will typically experience numerical instability. For both timing and stability concerns, this simplistic approach does not work for large matrices.

The landmark 1978 paper by Moler and Van Loan [4] entitled "Nineteen dubious ways to compute the exponential of a matrix", and its 2003 update by the same authors [5], review and explore a wide range of algorithms, none of which constitutes a good solution in all cases. Many algorithms suffer from numerical instability, especially when the entries of A are large in magnitude. In terms of numerical complexity, there are $\mathcal{O}(n^4)$ methods, as well as $\mathcal{O}(n^3)$ methods under certain conditions, but the resulting matrix is often full even if A is sparse, and there is no good way to benefit from sparsity, making this extremely expensive for large systems. In short, it is practically impossible to compute the exponential for very large matrices, in general. However, in specific problem domains, there may be reliable approximations that can do the job.

Given the size of the problems we deal with today, with billions of nodes in the chip power grid, even the best classical methods [4], [5] are not applicable. It is prohibitively expensive to run $\mathcal{O}(n^3)$ and $\mathcal{O}(n^4)$ algorithms on such problems, when sparsity cannot be leveraged. Even for single interconnect trees, these methods are not acceptable, and must be adapted using any information that may be available for the problem domain. There is currently no published work that provides a fast, efficient and accurate method for computing current constraints for chip safety from electromigration, or the matrix exponential of the system matrix under consideration - as far as we know. The method we propose reduces the matrix exponential based on Arnoldi's algorithm, which we briefly present in the next section.

D. Arnoldi's Algorithm

Arnoldi's algorithm [6] was first developed as an application of the method of minimized iterations in 1951. Although the original purpose of this algorithm was to reduce a dense matrix to upper Hessenberg[1] form, Arnoldi hinted at its potential use to approximate the largest eigenvalues of a matrix. The algorithm was later refined by Saad [7] and others, and has since been used to compute eigenelements of large unsymmetric matrices.

Given a general $n \times n$ matrix A, and a starting unit-norm vector v_1, Arnoldi's algorithm builds an orthogonal basis matrix Q of the Krylov subspace $\mathcal{K}_k \equiv \text{span}\{v_1, Av_1, A^2v_1, ..., A^{k-1}v_1\}$. By considering the Arnoldi method as a Rayleigh-Ritz projection on a Krylov subspace, it is known that the Ritz values in $\mathcal{K}_k$ and their corresponding Ritz vectors are good approximations for some eigenpairs of A.

The algorithm starts with the vector v_1 and a desired parameter $k < n$ that corresponds to the size of the reduced system. The first column of Q is v_1; as for the other columns $v_2, \cdots, v_k$, they are iteratively computed according to the recurrence relation

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=1}^{j} h_{ij}v_i, \tag{11}$$

where the h_{ij} are chosen such that each v_{j+1} is orthogonal to all previously found v_i $(i = 1, ...j)$ and $||v_{j+1}|| = 1$.

In [8, §6.2.1], Saad gives the basic Arnoldi algorithm implementing relation (11) with the orthonormality constraint, reproduced in Algorithm 1.

Algorithm 1 Basic Arnoldi Algorithm

```
1: procedure ARNOLDI(A, n, k)
2:     Choose a vector v_1 of norm 1
3:     for j = 1, 2, ..., k − 1 do
4:         h_ij = ⟨Av_j, v_i⟩,    i = 1, 2, ..., j
5:         w_j = Av_j − Σ_{i=1}^{j} h_ij v_i
6:         h_{j+1,j} = ||w_j||_2,    if h_{j+1,j} = 0 stop
7:         v_{j+1} = w_j / h_{j+1,j}
```

It was proved in [8, §6.2.1] that the collection of vectors $\{v_1, ..., v_k\}$ which form the columns of Q are a basis for the Krylov subspace $\mathcal{K}_k$ spanned by $\{v_1, Av_1, ..., A^{k-1}v_1\}$. The following relation is also proved, where H refers to the $k \times k$ upper Hessenberg matrix whose entries are the h_{ij} computed in Algorithm 1,

$$Q^*AQ = H \tag{12}$$

In the general case where $k < n$, the columns of Q are orthonormal. If we let Q^* be the conjugate transpose of Q, this translates to

$$Q^*Q = I. \tag{13}$$

Since Arnoldi's method stops before returning n columns of Q, it produces the approximation

$$A \approx Q^*HQ. \tag{14}$$

It is important to note that the matrix A is not necessarily well-approximated by the expression in (14). However, it is accepted in the literature that some of the k eigenvalues and eigenvectors of H constitute good approximations for some eigenpairs of A [8], with the approximation becoming better with increasing values of k. This makes Arnoldi's algorithm attractive if one is only interested in a few eigenpairs of A. Since H is a matrix of modest size $k \times k$, where k is chosen to be much smaller than n, and because of its Hessenberg structure, its eigenvalues can easily be computed with a procedure such as a sparse QR algorithm. Although the quality of the approximation of the eigenvalues seems to be mostly based on empirical observations, theoretical results for convergence are provided in [7]. In short, eigenvalues that are "extreme and isolated" will be well-approximated, whereas eigenvalues that are in the innermost part of the spectrum will have large errors. In practice, it has been observed that if the Arnoldi parameter k is chosen to be large enough,

[1] A square matrix H is said to be an upper Hessenberg matrix if it has only zero entries below the first diagonal, i.e., if $h_{ij} = 0$ for all $i > j + 1$.

a portion of the eigenvalues, corresponding to the largest-magnitude eigenvalues of A, will be well-approximated by the eigenvalues of H (with well-approximated eigenvectors), whereas the rest of the eigenpairs will typically be far from the actual eigenpairs of A.

III. Efficient Current Constraints Generation

We now go over our approach for computing the current constraints for a given interconnect tree, i.e., the vector s and the matrix K. In order to compute the former, we derive an expression based on some properties of the matrix exponential which makes the computation straightforward, and independent of the matrix exponential itself. The bottleneck of the procedure is the computation of K, which is intricately tied to that of the matrix exponential. We run Arnoldi's algorithm to generate a projection matrix, which is then used to project the stress equation onto a reduced subspace of dimension $k \ll n$. The reduced matrix exponential is then efficiently computed with a procedure based on eigenvalues and eigenvectors. Finally, the system is projected back to the original system of dimension n.

A. Reduction of the System Matrix A

The first step of our approach is to reduce the system matrix A to an approximate matrix of smaller dimension, which still captures its action on the stress evolution. We perform this reduction with Arnoldi's algorithm, in order to preserve some of the eigenvalues of $A = -C^{-1}G$. Since C is a diagonal matrix with strictly positive diagonal entries, it is symmetric positive definite (SPD), and its inverse C^{-1} exists and is also SPD. As for the matrix G, it is the Laplacian [9], [10] of a connected graph, so it is symmetric positive semi-definite (SPSD), all its eigenvalues are real and non-negative, and it has a simple eigenvalue at 0 with corresponding eigenvector $\mathbb{1}$, where $\mathbb{1}$ is the vector whose every entry is 1. Since the product of two SPSD matrices has non-negative real eigenvalues [11, Theorem 7.5], then all the eigenvalues of A are real and non-positive. In the stress equation (7), A appears in the exponential term, which means that its large eigenvalues (in absolute value) will produce a very fast transient response. However, these transients are not significant when dealing with a slow effect like electromigration. The fast components of the dynamic response dissipate so quickly that their impact on the very slow EM stress evolution over time is negligible. They are effectively filtered out due to the slow dynamics of system, which are themselves determined by the eigenvalues of small magnitude. We therefore choose to reduce the matrix in a way that preserves its smallest eigenvalues (in absolute value).

Arnoldi's algorithm produces an upper Hessenberg matrix H whose eigenvalues approximate the large-magnitude eigenvalues of the input matrix A. In order to get an approximation based on the small-magnitude eigenvalues, we use the shift-and-invert Arnoldi algorithm [12]. It consists of preprocessing the input matrix A by shifting it (in order to eliminate its 0 eigenvalue, which is due to the 0 eigenvalue of G) and inverting it. In other words, the input of the Arnoldi algorithm becomes

$$\hat{A} \triangleq (A - \delta I)^{-1}. \tag{15}$$

The matrix $\hat{A}$ is fed into the Arnoldi algorithm, which outputs two matrices $\hat{Q}$ and $\hat{H}$. If we let the $k \times k$ matrix $\bar{H}$ be

$$\bar{H} \triangleq \hat{H}^{-1} + \delta I, \tag{16}$$

then it can be shown [13] that the eigenvalues of $\bar{H}$ approximate the k smallest-magnitude eigenvalues of A (and the corresponding eigenvectors are the same as those of $\hat{H}$). This is due to the properties of shift and inversion and their effect on the eigenvalues and eigenvectors of a matrix. This leads to the key approximation

$$\bar{H} \approx \hat{Q}^* A \hat{Q}, \tag{17}$$

based on the small-magnitude eigenvalues of A. Note that in practice, no inversion is required because we do not explicitly compute $\hat{A}$. Instead, we use a modified version of Arnoldi's algorithm [12] with the shifted A matrix as input, and whose only overhead is the computation of an LU factorization of A. The details are in [13].

B. Projection of the System

In the previous section, we saw that the system matrix A can be projected onto a smaller subspace of dimension $k \times k$ using Arnoldi's algorithm. We will now see how the outputs $\hat{H}$ and $\hat{Q}$ of Arnoldi's algorithm can be used to compute the matrix exponential in the reduced subspace.

Recall that the system under consideration gives the stress evolution over time (4). Since the input currents i are constant, the system (4) can be equivalently expressed by defining

$$z(t) = \sigma(t) - G^+ HMDM^+ i, \tag{18}$$

and substituting $\sigma(t) = z(t) + G^+ HMDM^+ i$ back into (4) to get

$$\dot{z}(t) = -C^{-1}Gz(t) = Az(t) \tag{19}$$

whose solution is

$$z(t) = e^{At} z_0. \tag{20}$$

By using the $n \times k$ matrix $\hat{Q}$ generated by Arnoldi's method as a projection matrix, we can project this homogeneous system onto a Krylov subspace of dimension k. Assuming that the state vector z is constrained to a low-dimensional subspace spanned by the linearly independent columns of $\hat{Q}$, we have

$$z(t) = \hat{Q}\hat{z}(t), \tag{21}$$

where $\hat{z}(t)$ is a $k \times 1$ vector, from which

$$\hat{z}(t) = \hat{Q}^* z(t), \tag{22}$$

which is obtained by pre-multiplying both sides by Q^* and then using (13) to replace $\hat{Q}^*\hat{Q}$ by I. We now perform a change of variable by replacing z in (19) by $\hat{Q}\hat{z}$, then pre-multiply by $\hat{Q}^*$ to get

$$\dot{\hat{z}}(t) = \hat{Q}^* A \hat{Q} \hat{z}(t). \tag{23}$$

By using the approximation in (17), this becomes

$$\dot{\hat{z}}(t) \approx \bar{H}\hat{z}(t), \tag{24}$$

whose solution is

$$\hat{z}(t) \approx e^{\bar{H}t}\hat{z}_0. \tag{25}$$

Note that $\bar{H}$ is a $k \times k$ matrix, which is much smaller than the original system matrix A of size $n \times n$. The matrix exponential in this reduced subspace is therefore much easier to compute than the original one. After the system is solved in the reduced subspace, and we obtain (25), we can recover the original variable z through (21) and (22), as

$$z(t) \approx \hat{Q}e^{\bar{H}t}\hat{Q}^* z_0. \tag{26}$$

Now going back to the σ variable by using (18) and rearranging the terms gives

$$\sigma(t) \approx \hat{Q}e^{\bar{H}t}\hat{Q}^*\sigma_0 + (I - \hat{Q}e^{\bar{H}t}\hat{Q}^*)G^+HMDM^+i. \tag{27}$$

Since we are only interested in the time point $t = T$, comparing to (4), the resulting approximate current constraints are based on

$$K \approx (I - \hat{Q}e^{\bar{H}T}\hat{Q}^*)G^+HMDM^+ \tag{28}$$

$$s \approx \sigma_{\text{crit}} - \hat{Q}e^{\bar{H}T}\hat{Q}^*\sigma_0. \tag{29}$$

It is clear that the essence of our approach is in fact

$$e^{AT} \approx \hat{Q}e^{\bar{H}T}\hat{Q}^*. \tag{30}$$

This highlights the fact that the computation of the matrix exponential is simplified by computing the exponential of a reduced matrix, which is then projected back to the original space.

C. Computation of the Reduced Matrix Exponential

The application of the Arnoldi-based approximation reduces the computation of an $n \times n$ matrix exponential e^{AT} to that of a $k \times k$ matrix exponential $e^{\bar{H}T}$, which is still a nontrivial task. We now explain the strategy we adopt for this computation.

Recall that $\bar{H} = \hat{H}^{-1} + \delta I$. Instead of explicitly computing $\bar{H}$ and then finding its matrix exponential (which would require a matrix inversion, and would lose the Hessenberg structure of $\hat{H}$), we first find the eigenvalues of $\hat{H}$ and their corresponding eigenvectors - which is easier than for a general matrix, because it is upper Hessenberg. From these, we can readily obtain the eigenvalues and eigenvectors of $\bar{H}$, and the computation of the matrix exponential becomes straightforward, as we will now show.

Note that, since $\hat{H}$ is approximating the eigenvalues of the shifted and inverted version of A (which is diagonalizable because it is the product of two SPSD matrices), it is fair to assume that $\hat{H}$ is diagonalizable (and its eigendecomposition exists).

Theorem 1. *Let $\bar{H} = (\hat{H}^{-1} + \delta I)$ as defined in (16), and let $[V, \Lambda]$ be an eigendecomposition of $\hat{H}$. Then, $e^{\bar{H}T} = Ve^{\bar{\Lambda}T}V^{-1}$ where $\bar{\Lambda} = \Lambda^{-1} + \delta I$.*

Proof. $[V, \Lambda]$ is an eigendecomposition of $\hat{H}$. Then

$$\hat{H} = V\Lambda V^{-1}. \tag{31}$$

Formally, if $\hat{H}$ is of size $k \times k$, then V is the $k \times k$ matrix whose i^{th} column is the eigenvector v_i of $\hat{H}$, and Λ is the $n \times n$ diagonal matrix whose diagonal elements are the corresponding eigenvalues $\Lambda_{ii} = \lambda_i$. Once these two matrices are available, an eigendecomposition of $\bar{H}$ can be easily obtained, as shown in the following

$$\hat{H} = V\Lambda V^{-1} \tag{32}$$

$$\hat{H}^{-1} = V\Lambda^{-1}V^{-1} \tag{33}$$

$$\hat{H}^{-1} + \delta I = V\Lambda^{-1}V^{-1} + \delta I \tag{34}$$

$$\hat{H}^{-1} + \delta I = V(\Lambda^{-1} + \delta I)V^{-1} \tag{35}$$

If we let

$$\bar{\Lambda} = \Lambda^{-1} + \delta I, \tag{36}$$

then the eigendecomposition of $\bar{H}$ can be expressed as

$$\bar{H} = V\bar{\Lambda}V^{-1}. \tag{37}$$

Note that $\bar{\Lambda}$ can be easily obtained from Λ, since the inversion of the diagonal $k \times k$ matrix Λ is a simple $\mathcal{O}(k)$ procedure corresponding to the inversion of each diagonal entry. The matrix $\bar{\Lambda}$ is also a diagonal matrix, and it contains the approximate k smallest-magnitude eigenvalues of the original matrix A, whose corresponding eigenvectors are the same as Λ.

Once the eigendecomposition in (37) is available, the computation of the matrix exponential becomes a straightforward procedure whose bottleneck is a matrix-matrix multiplication. Since V is an orthogonal matrix, and $\bar{H} = V\bar{\Lambda}V^{-1}$, the matrix exponential can be expressed as

$$e^{\bar{H}T} = Ve^{\bar{\Lambda}T}V^{-1}. \tag{38}$$

which is a well-known result in linear algebra (and may be derived from the definition of the matrix exponential as an infinite sum). The matrix exponential $e^{\bar{\Lambda}T}$ is obtained by exponentiating each diagonal entry of the matrix of eigenvalues, pre-multiplying it by the matrix of eigenvectors, and post-multiplying it by its inverse. □

To sum up, once an eigendecomposition of $\hat{H}$ is available, we can obtain an eigendecomposition of $\bar{H}$ in $\mathcal{O}(k)$ time, and we can then compute the matrix exponential of $\bar{H}$ (multiplied by the scalar T) with two simple matrix-matrix multiplications. This leads to the approximation

$$K \approx (I - \tilde{Q}Ve^{\bar{\Lambda}T}V^{-1}\tilde{Q}^*)G^+HMDM^+. \tag{39}$$

D. Approximation of s

Although the approximation from (29) can be used to efficiently compute the s vector, we choose to compute it in a different, and much faster way. We assume that the chip under consideration is starting from a "resting state", which means that enough powered-down time has passed so that the initial stress is the same everywhere, i.e., $\sigma_0 = \sigma_i \mathbb{1}$, where σ_i is a

scalar. We will show that under this assumption, the vector s is equal to $\sigma_{crit} - \sigma_0$.

Recall that G is a conductance matrix, which means that the sum of each one of its rows is 0. Formally, $G\mathbb{1} = 0$. Now, let us look at the product $e^{AT}\mathbb{1} = e^{-C^{-1}GT}\mathbb{1}$. If we use the definition of the matrix exponential as an infinite sum, we can see that each term, except the initial $I\mathbb{1}$ term, will end with the product $G\mathbb{1}$. This means that each term in the infinite sum is equal to zero, except the first one. In other words, $e^{AT}\mathbb{1} = I\mathbb{1} = \mathbb{1}$.

We now use this property to compute the matrix exponential. Since the initial stress is the same everywhere, we can write $s = \sigma_{\text{crit}} - e^{AT}\sigma_0$ as $\sigma_{\text{crit}} - \sigma_i e^{AT}\mathbb{1}$. It is then easy to see that $s = \sigma_{\text{crit}} - \sigma_i\mathbb{1}$, leading to the final equation

$$s = \sigma_{\text{crit}} - \sigma_0. \tag{40}$$

IV. Simulation results

We developed a C++ approach which implements the algorithm presented in Section III. All computations were performed on a hyperthreaded 12-core 3GHz Linux machine with 128GB of RAM. Since the input matrices G, C, M, H and D are highly sparse, we used the CHOLMOD package [14] from SuiteSparse [15] to store them and perform operations on them. Finally, the C++ template library Eigen [16] was used to find the eigenvalues, as needed in Section III-C. We judge our approximation based on its accuracy and runtime, when compared to the exact computation. Unless otherwise stated, the computations are carried out for a target lifetime of $T = 10$ year, with a constant Arnoldi parameter k of 100, and a shift δ of 10^{-15} for the shift-and-invert Arnoldi procedure.

A. Accuracy Results

First, we present the accuracy results, which are central to the quality of the approximation. The approximate K matrix is generated according to (39), using the C++ procedure outlined in Section III. The approximate K matrix is compared to the exact K, which is computed by finding the exact matrix exponential. Since developing an algorithm for exact matrix exponential computation in C++ is an entirely separate task, we chose to perform the exact computation in MATLAB®, in two different ways:

- The first one using the built-in matrix exponential command **expm** provided by MATLAB®.
- The second one using **fastExpm**, a function initially written by I. Kuprov [17] [18] in 2011 and later adapted by F. Mentink-Vigier [19]. It leverages the sparsity of the matrix by setting very small entries to 0, and uses scaling, Taylor series and squaring to preserve matrix sparsity over the different iterations.

In practice, **expm** is very slow, and it is thus not scalable for large matrices, whereas **fastExpm** makes for a fairer comparison with our approximation. Note that **expm** and **fastExpm** produce the exact same output, so accuracy is not affected by the use of the faster user-defined function for comparison.

The first error metric is the normalized root-mean-square (RMS) error, which is effectively the standard deviation of the residuals (prediction errors) between each approximate entry of K and its exact counterpart. It is calculated by summing the squares of the residuals of all the entries, dividing the sum by the number of entries, and then taking the square root of the result. The RMS error is then normalized by dividing by the average value of all the entries in the exact K matrix. The second error metric we report is the normalized maximum error. This is simply the maximum absolute value of the difference between an approximate value and its exact counterpart, divided by the average value of all the entries in the exact K. Finally, we also report the normalized average value of the residuals, i.e., the normalized average prediction error. It is obtained by taking the mean of all the residuals, and then dividing this value by the average value of all the entries in the exact K matrix.

For all three error metrics, normalization is essential because the values of K are often of the order of magnitude of 10^6 or larger. The results are compiled in Table I for increasing values of m.

TABLE I
ACCURACY MEASURES FOR THE APPROXIMATE CURRENT CONSTRAINTS MATRIX K FOR SAFETY AT $T = 10$ YEARS AND $k = 100$.

Junction count m	Node count n	Normalized RMS Error	Normalized Maximum Error	Normalized Average Prediction Error
300	5981	0.06%	0.37%	0.04%
700	13,981	0.31%	4.15%	0.19%
1500	29,981	1.14%	27.46%	0.52%

From Table I, we see that the errors increase with circuit size, but no circuit has a normalized RMS error larger than 1.2%, indicating very good agreement between the approximate and exact matrices. This shows that keeping a very small number of the smallest eigenvalues (in magnitude) is enough to capture the action of the matrix exponential on the product G^+HMDM^+, for circuits of reasonable size. Note that the decrease in accuracy with circuit size was expected, because we are keeping the size of the reduced subspace constant, even for increasingly large matrices.

B. Timing Results

We now evaluate the speed of the approximation by comparing the time it takes to compute the approximate K matrix in C++, and the exact K in MATLAB® (by the two different methods mentioned in Section IV-A), for different circuits with increasing values of m.

Fig. 1 shows the measurements for circuits having between 100 and 900 junctions (n between 1981 and 17,981), with the interpolated polynomial lines. Fig. 2 shows measurements for the approximation for circuits having m up to 5500 ($n = 109{,}981$), with the interpolated line, and the extrapolations of the lines from Fig. 1 for the exact computations. The largest circuits for which we actually measured the runtime of the

exact solution had $m = 900$ ($n = 17{,}981$) for the built-in **expm** and $m = 1500$ ($n = 29{,}981$) for **fastExpm**. For larger circuits, inherent MATLAB® limitations and very large simulation times make it impractical to compute the exact solution; and for these same reasons, the largest circuit tested for accuracy in Section IV-A has $m = 1500$ junctions ($n = 29{,}981$).

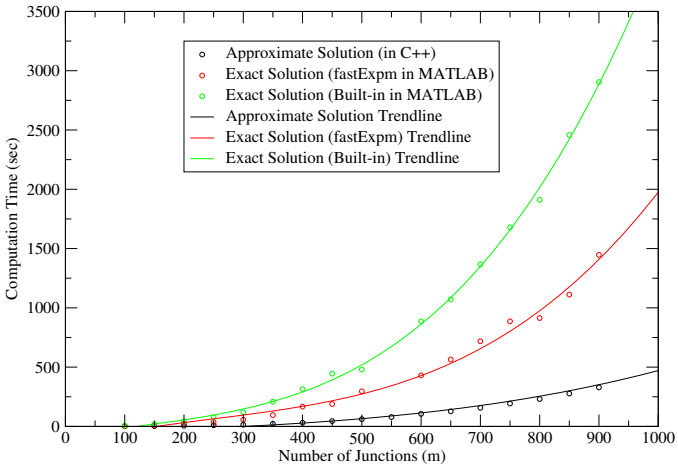

Fig. 1. Runtime of the exact and approximate methods for computing K for different circuits with m up to 900 (n up to 17,981), and $k = 100$.

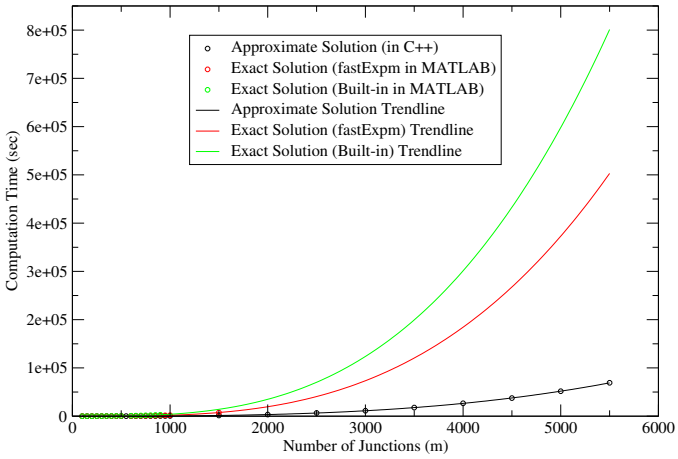

Fig. 2. Runtime of the exact and approximate methods for computing K (with extrapolated lines) for m up to 5500 (n up to 109,981), and $k = 100$.

In addition to the figures, the timing results for some circuits with up to $m = 900$ junctions are shown in Table II. Clearly, computing the exact solution takes significantly more time, and is inherently limited by the capabilities of MATLAB®, whereas the approximate solution can be computed for much larger circuits, with up to 5500 junctions.

From Figs. 1 and 2, we can see that the approximation offers a considerable speedup compared to both exact methods. For instance, for $m = 900$, the approximation took 5 min 29 sec, the computation of the exact K using **fastExpm** took 24 min 7 sec, and the exact K using **expm** took 48 min 24 sec. For the largest test case ($m = 5500$), the approximate computation took 19 hrs 13 min. Based on the extrapolated polynomial lines, the exact computation would take 4 days 23 hrs with **fastExpm** and 8 days 23 hrs with **expm**.

TABLE II
RUNTIME OF THE APPROXIMATE (IN C++) AND EXACT (IN MATLAB WITH THE BUILT-IN FUNCTION AND FASTEXPM) CURRENT CONSTRAINTS COMPUTATION, FOR CIRCUITS WITH 100 JUNCTIONS UP TO 900 JUNCTIONS, FOR SAFETY AT $T = 10$ YEARS AND WITH $k = 100$.

Junction count m	Approximate Solution runtime (in C++)	Exact solution runtime (**fastExpm** in MATLAB)	Exact solution runtime (built-in in MATLAB)
100	1 sec	3 sec	6 sec
200	5 sec	21 sec	43 sec
300	14 sec	56 sec	2 min
400	31 sec	2 min 47 sec	5 min 15 sec
500	1 min 1 sec	4 min 56 sec	8 min
600	1 min 46 sec	7 min 10 sec	15 min
700	2 min 37 sec	12 min	22 min
800	3 min 51 sec	15 min	32 min
900	5 min 29 sec	24 min	48 min

C. Choice of the Arnoldi Parameter k

In this section, we explain our choice of the Arnoldi parameter k. We used a fine-tuning process by running simulations on different circuits while varying k, and monitoring the accuracy measures.

The parameter k is one of the most important "knobs" of our approximation: it determines the size of the reduced subspace, namely, the dimension of the reduced matrix whose exponential will have to be computed. It also determines how much "information" will be retained by the approximation (by specifying how many eigenpairs should be approximated), and by extension, how much will be dropped. The smaller k is, the faster the approximation will be, but also the worse the overall accuracy. Choosing an appropriate value for k is thus a tradeoff between speed and accuracy: we wish to choose the smallest value of k which provides an accurate enough approximation.

Fig. 3 shows the normalized RMS error and the normalized average prediction error for a circuit with $m = 700$ junctions ($n = 13{,}981$). In both cases, k is varied between 40 and 400. For both error metrics, we see downward curves which start at a relatively high error level (albeit still quite low - less than 1.2% in all cases), and reach very low error values. Note that values of k less than 40 are not shown in the graph for clarity, because they produce much higher errors. For instance, for $m = 700$ and $k = 20$, the normalized RMS error is 4.5 %.

For values of k greater than 50, it seems that the error is already quite low for both circuits tested, and that there is no point in increasing k above a certain point, because the gain in accuracy is minimal and it causes a significant increase in compute time. Note, however, that for a specific value of k, the errors seem to increase with circuit size. For instance, for $k = 200$, the normalized RMS error was 0.0214% for $m = 300$ ($n = 5981$), and 0.1286% for $m = 700$ ($n = 13{,}981$). For this reason, one of the options we considered

was to vary k with m, so that the accuracy stays relatively constant no matter the circuit size. In fact, this is the strategy we recommend for very large circuits having tens of thousands of junctions.

In our case, however, circuit size was capped at about 5500 junctions, and we chose to have a constant value of k equal to 100, since it provided reasonable accuracy for all circuit sizes: the normalized RMS error was 0.0577% for $m = 300$, 0.3138% for $m = 700$, and 1.1444% for $m = 1500$. Since the RMS error seems to be increasing linearly with circuit size, a simple linear extrapolation gives an error of 4.685% for $m = 5500$ ($n = 109{,}981$). To put things into perspective, this is still an error of less than 5%, for a system matrix of size $109{,}981 \times 109{,}981$, and a reduced matrix of size 100×100.

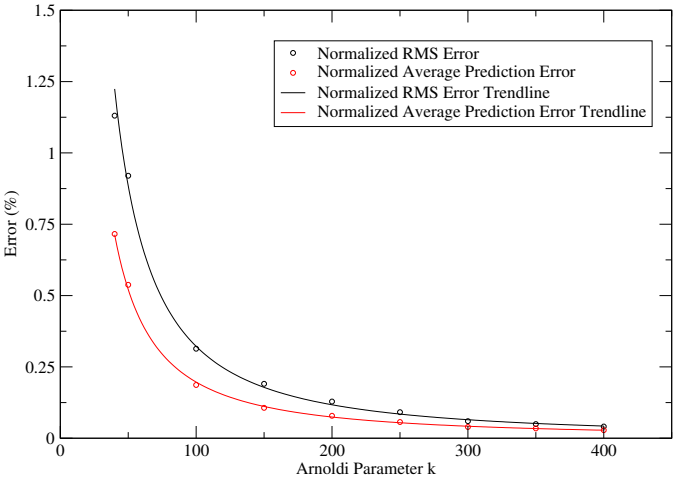

Fig. 3. Plot of the normalized RMS and average prediction errors between the approximate and exact current constraints, for a circuit with $m = 700$ junctions ($n = 13{,}981$), and for different values of the Arnoldi parameter k.

V. Conclusion

Power grid reliability is a major concern in chip design, and it can be significantly affected by electromigration. The effects of electromigration threaten the integrity of the chip's power grid and its ability to provide the desired voltages to the underlying circuitry. In this work, we have developed a tool to guarantee chip safety from voids until a target lifetime, by efficiently generating constraints on the underlying source currents. The computation of the constraints relies on a model order reduction scheme based on Arnoldi's algorithm, in order to efficiently compute a large matrix exponential, which is the bottleneck of the constraints generation process. Our tool can handle much larger test cases compared to existing methods, with error metrics lower than 5% in all test cases.

References

[1] S. P. Hau-Riege and C. V. Thompson, "Experimental characterization and modeling of the reliability of interconnect trees," *Journal of Applied Physics*, vol. 89, no. 1, pp. 601–609, Jan. 2001.

[2] C. G. Chaanin, "Generating constraints for electromigration safety for chip power grids," MASc Thesis, University of Toronto, 2021.

[3] F. N. Najm, "Equivalent circuits for electromigration," *Microelectronics Reliability*, vol. 123, p. 114200, 2021.

[4] C. Moler and C. Van Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," *SIAM Review*, vol. 45, no. 1, pp. 3–49, 2003. [Online]. Available: http://link.aip.org/link/?SIR/45/3/1

[5] ——, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," *SIAM Review*, vol. 45, no. 1, pp. 3–49, 2003. [Online]. Available: https://doi.org/10.1137/S00361445024180

[6] W. E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem," *Quarterly of Applied Mathematics*, vol. 9, no. 1, pp. 17–29, 1951. [Online]. Available: http://www.jstor.org/stable/43633863

[7] Y. Saad, "Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices," *Linear Algebra and its Applications*, vol. 34, pp. 269–295, 1980. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002437958090169X

[8] ——, *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, 1992.

[9] P. Van Mieghem, K. Devriendt, and H. Cetinay, "Pseudoinverse of the Laplacian and best spreader node in a network," *Physical Review E*, vol. 96, no. 3, p. 032311, Sep. 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.96.032311

[10] N. K. Vishnoi, "Lx = b," *Foundations and Trends® in Theoretical Computer Science*, vol. 8, no. 1–2, pp. 1–141, 2013. [Online]. Available: nowpublishers.com/article/Details/TCS-054

[11] F. Zhang, *Matrix Theory: Basic Results and Techniques*. Springer New York, 2011. [Online]. Available: https://books.google.ca/books?id=iWMYmL801vQC

[12] Z. Jia and Y. Zhang, "A refined shift-and-invert Arnoldi algorithm for large unsymmetric generalized eigenproblems," *Computers and Mathematics with Applications*, vol. 44, no. 8, pp. 1117–1127, 2002.

[13] C. Feghali, "Power grid safety under electromigration," MASc Thesis, University of Toronto, 2023.

[14] Y. Chen, T. Davis, W. Hager, and S. Rajamanickam, "Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate." *ACM Trans. Math. Softw.*, vol. 35, 01 2008.

[15] T. Davis *et al.*, "Suitesparse version 5.13.0," 2022. [Online]. Available: https://github.com/DrTimothyAldenDavis/SuiteSparse

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

[17] H. J. Hogben, M. Krzystyniak, G. T. P. Charnock, P. Hore, and I. Kuprov, "Spinach–a software library for simulation of spin dynamics in large spin systems." *Journal of magnetic resonance*, vol. 208 2, pp. 179–94, 2011.

[18] I. Kuprov, "Diagonalization-free implementation of spin relaxation theory for large spin systems," *Journal of Magnetic Resonance*, vol. 209, no. 1, pp. 31–38, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1090780710003964

[19] F. Mentink-Vigier, "Fast exponential matrix for matlab (full/sparse), fastexpm," https://github.com/fmentink/fastExpm/releases/tag/1.0, 2023.