UNIVERSITY OF CALIFORNIA
RIVERSIDE

Modeling and Simulation Methods for VLSI Interconnect Reliability Focusing on Time Dependent Dielectric Breakdown

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Shaoyi Peng

March 2021

Dissertation Committee:

Dr. Sheldon Tan, Chairperson
Dr. Hyoseung Kim
Dr. Daniel Wong

The Dissertation of Shaoyi Peng is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I wish to thank many people who gave me the support and help through the journey of my PhD studies.

First and foremost, I want to express my gratitude to my knowledgeable advisor, Dr. Sheldon Tan, without whose help, I would not have been here. His experience and mentorship is beyond helpful while I was carrying out my research, and also while publishing my research. Our discussions about work and career are invaluable to me.

I would also like to thank my committee members Dr. Hyoseung Kim and Dr. Daniel Wong for their guide to my research, especially during my first year when I started to explore the areas of research.

I am also grateful to all my fellow labmates. Especially, I want to thank Taeyoung Kim, Hengyang Zhao, Chase Cook, Zeyu Sun, Han Zhou, Sheriff Sadiqbatcha, Wentian Jin, Jinwei Zhang, Shuyuan Yu and Yibo Liu. The mentorship and discussions we had are treasurous during these years of study. Furthermore, I cannot imagine going through these years without your company. You are the family of mine in this country.

The content of this thesis is reprinted or rewritten from these published materials:

- Shaoyi Peng, Han Zhou, Taeyoung Kim, Hai-Bao Chen, and Sheldon X-D Tan. "Physics-based compact TDDB models for low-k beol copper interconnects with time-varying voltage stressing". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2017 (Chapter 2)

- Shaoyi Peng, Ertugrul Demircan, Mehul D. Shroff, and Sheldon X-D. Tan. "Full-chip wire-oriented back-end-of-line TDDB hotspot detection and lifetime analysis". Integration, 2020 (Chapter 3)
- Shaoyi Peng, Wentian Jin, Liang Chen, and Sheldon X-D Tan. "Data-Driven Fast Electrostatics and TDDB Aging Analysis". Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD (Chapter 4)
- Shaoyi Peng, and Sheldon X-D. Tan. "GLU3.0: Fast GPU-based Parallel Sparse LU Factorization for Circuit Simulation". IEEE Design & Test, 2020 (Chapter 5)

To my parents for all the love and support.

ABSTRACT OF THE DISSERTATION

Modeling and Simulation Methods for VLSI Interconnect Reliability Focusing on Time Dependent Dielectric Breakdown

by

Shaoyi Peng

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, March 2021
Dr. Sheldon Tan, Chairperson

Time dependent dielectric breakdown (TDDB) is one of the important failure mechanisms for Copper (Cu) interconnects that are used in VLSI circuits. This reliability effect becomes more severe as the space between wires is shrinking and low-k dielectric materials (low electrical and mechanical strength) are used. There are many studies and theories focusing on the physics of it. However, there is limited research from the electronics design automation (EDA) perspective on this topic, aiming to evaluate, or alleviate it from the perspective of designing a VLSI chip. This thesis compiles several studies into evaluating TDDB on the circuit level, and engineering methods that help the evaluation. The first work extends the study of a published physics model on simplified yet practical cases. It simplifies the calculation of lifetime by deriving an analytic solution and applying fitting methods. The second study proposes a new way to evaluate lifetime of a chip by extending the models of simple interconnect structures to the complete chip. This method is more robust as it focuses more on a complete chip. However, heavy dependence of finite element method (FEM) makes the flow very slow. The third study adopts machine learning methods

to accelerate this slow evaluation process. The proposed method is also applicable to other similar electrostatics applications. Last but not least, the fourth study focuses on a GPU based LU factorization algorithm, which, on a broader aspect, is a universal numerical algorithm used in many different simulation applications, which can be helpful to TDDB evaluations as it can be used in FEM.

Contents

List of Figures

List of Tables

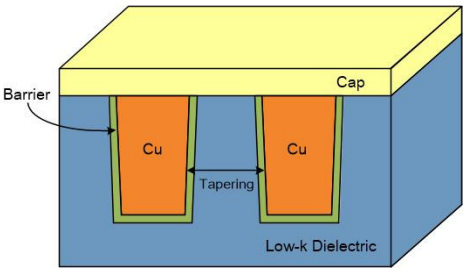
Chapter 1

Introduction

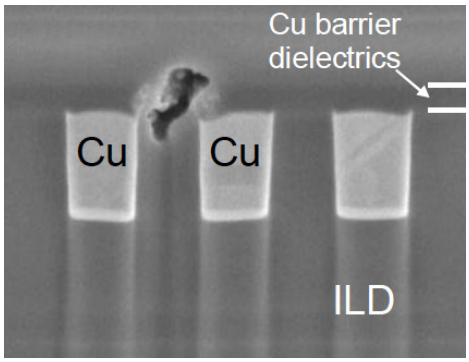
1.1 Time Dependent Dielectric Breakdown for VLSI Interconnect

Time Dependent Dielectric Breakdown (TDDB) is the physical phenomenon that dielectric breaks down with time when the dielectric deposited under a field becomes weaker than the material breakdown strength. It creates a short circuit and is thus fatal to the VLSI circuit. TDDB has become a serious dielectric reliability concern and failure mechanism for back end of line (BEOL) interconnects, so accurate and fast modeling and estimation of TDDB failure become important.

Traditionally, TDDB is a major concern only for the gate oxide of CMOS devices. With the advances of technology scaling, middle-of-line (MOL) TDDB and BEOL TDDB have also become important reliability mechanisms. MOL TDDB is breakdown between the gate and contact and is often modeled together with gate oxide TDDB as they can be



(a) The cross-section of copper/low-k dielectric structure



(b) The cross-section SEM image after TDDB failure, courtesy of [60]

Figure 1.1: Structure of two copper interconnect wires and the IMD, where TDDB occurs

modeled by extra resistors connecting transistor nodes [70, 8]. However, with aggressive technology scaling accompanied by the employment of new low-k and even porous materials [17], TDDB becomes a serious reliability concern and failure mechanism for BEOL interconnects [47, 34, 5, 67]. The dielectric breakdown is caused by the conducting path formation through the inter-metal dielectric (IMD) oxide between metal wires from electron tunneling current. Finally, a significant large leakage current occurs and results in the chip operation failure. This is shown in the SEM image in Fig. 1.1, along with a descriptive figure showing the structure of the two metal wires and the IMD in between. For BEOL, TDDB effects can be seen at the inter-layer dielectric (ILD) between two metal layers and also IMD between metal wires in the same layer of the low-k dielectrics. In general, due to the fact that ILD space is larger than that of IMD, TDDB in IMD is more likely to occur so more research focus is put on it.

Many researches have been done about the physics of TDDB, and thus many TDDB models have been proposed based on different proposed breakdown mechanisms,

the most well known ones being E, $1/E$, and $E^{1/2}$ models. The E-model models TDDB as a weak bond breakage due to thermochemical heating [44]. The $1/E$-model models it as high energy hole injection induced failure [9]. These models were initially introduced for gate oxides and later examined for the extension to the copper/low-k interconnects. The difference between gate oxides model and low-k BEOL dielectrics is the existence of metal ions in the former's interior. The $E^{1/2}$-model has been firstly proposed for metal-SiN-metal capacitors [2] and later examined in the low-k TDDB by assuming that the copper ion plays an important role in the dielectric breakdown [6, 60]. These studies use simple interconnect structures such as serpentine-comb or comb-comb to test the life time of test structures [67].

Recently some studies show that the migration of copper ions is not significant due to the existence of copper barrier (i.e. Ta). Instead, the barrier ions play an important role in ion migration [37, 43, 33]. Moreover, the breakdown depends on the conducting path formation between the two metal wires. This leads to the development of the electric path generation (EPG) model [33].

1.2 Sparse LU factorization

LU factorization represents the routine to factorize a full-rank square matrix A into the product of two triangular matrices: the lower triangular matrix L and upper triangular matrix U such that $A = LU$.

Sparse matrix is a special type of matrix, in which most of the elements are zero. Sparse matrices appear in many engineering and scientific computing applications, such as circuit simulation and finite element method (FEM), which itself is an important method

in TDDB analysis. The core of the computing or dominant computing of these applications is to solve the linear algebraic system, $Ax = b$, where matrix A and vector b are given and x is a vector to be solved. LU factorization is an important way to solve such problem. After finding the triangular matrices L and U through LU factorization, the solution x is then obtained by solving the two triangular matrices sequentially, which only has linear time complexity and is much cheaper than LU factorization.

1.3 Related works

1.3.1 TDDB

Besides the works on the physics of TDDB mentioned above, there also exists some other works focusing on evaluating the BEOL TDDB lifetime of a chip, which is the time a chip can function before a TDDB failure. They are more related to the studies conducted in this thesis.

A chip-scale TDDB lifetime simulator is proposed in [3]. In this work, interconnect geometries are grouped by spacing between wires and the total run length of each group is extracted. Next, lifetime of each group is scaled by total length based on experimental data of dielectrics of the same spacing. Furthermore, practical considerations such as variation in spacing and different temperature across the chip are also done. [4] extends this work to include other reliability effects such as electromigration (EM) and stress-induced voiding (SIV).

Besides proposing the EPG model, [33] also introduces a method based on a look-up table of high-risk interconnect shape patterns. The look-up table is built by simulating

the TDDB failures on the patterns. Also, fitting methods are used to cover different kinds of wires such as power, ground and signal.

The work presented in [35] models BEOL TDDB as additional resistors across dielectrics. The device level model is then used for the higher circuit and system level modeling and simulations.

1.3.2 Sparse LU factorization

There exists many earlier researches targeting sparse LU factorization. For instance, SuperLU_MT [22] is the multi-threaded parallel version of SuperLU for multi-core architectures. However, it is not easy to form super-node in some sparse matrix such as circuit matrix. KLU [20], which is specially optimized for circuit simulation, adopts Block Triangular Form based on Gilbert-Peierls (G/P) left-looking algorithm [27] and has become one of the standard algorithms in circuit simulation applications. The original KLU runs on single CPU core only. NICSLU [12, 13, 11] implements its parallel version on multi-core architecture by exploiting the column-level parallelism. UMFPACK [64] is implemented based on a multifrontal algorithm. PARDISO [57] is developed based on a left-right-looking algorithm.

Existing GPU based parallel LU factorization solvers mainly focus on dense matrices including [25]. There also exists a few sparse matrix LU factorization methods on GPU [14, 26]. But these works mainly convert the sparse matrices into many dense submatrices (blocks) and then solve them by dense matrix LU factorizations. However, such strategy may not work well for circuit matrices, which hardly have dense submatrices.

Parallel G/P left-looking algorithm on GPU has been explored first in [54, 10]. It exploits the column-level (called task-level) parallelism due to sparse nature of the matrix and vector-level parallelism in the sparser triangular matrix solving in the G/P method. However, the two loops in triangular matrix solving can't be completely parallelized (from line 4-8 in Algorithm 3) thus the G/P method is difficult for fine grain parallelization.

To mitigate this problem, He *et. al* proposed a hybrid right-looking sparse LU factorization on GPU, called GLU (GLU1.0) [30]. GLU keeps the benefits of the left-looking method for column-based parallelism and uses the same symbolic analysis routine. The difference is that it performs the submatrix update once one column is factorized, which is similar to the traditional right-looking LU method. However, GLU1.0 used a fixed scheme to allocate the GPU threads and memory, which limits its parallelism. Furthermore, the right-looking feature of GLU actually introduces new data dependency (called double-U dependency in this paper), which has been reported in GLU2.0 and [39]. Double-U dependency can lead to inaccurate results for some test cases. Detection of double-U dependency was added into GLU2.0 to fix this issue, which, however, incurred some performance degradation compared to GLU1.0. Recently, Lee *et al.* proposed an enhanced GLU2.0 solver [39], which considers the column count difference in different level, and exploits some advanced GPU features such as dynamic parallelism to further improve the GLU kernel efficiency. However, the fixed GPU threads and memory allocation method from GLU2.0 for each kernel launch is still used and limiting performance.

1.4 Contributions

The work presented in this thesis presents several contributions in the area of TDDB analysis for VLSI interconnect, and GPU based sparse LU factorization:

- The recently proposed physics based EPG model for interconnect TDDB is further studied. It is shown that the simplified 1D case can represent most cases pretty well. The analytic solution of the PDE of the model is derived. Then it is shown that the location of the minimum concentration, which is a key factor in this TDDB model, can be determined by the dominant terms and the TTF can be computed by using a few dominant terms. Furthermore, fitting methods are adopted to extend the proposed method to time-varying DC stressing voltages.
- A full chip TDDB lifetime and hotspot evaluation flow is proposed based on a new concept called TDDB Damage. It takes into account the layout effect brought by complex wire geometries and the length of the wires and dielectric. By partitioning the chip into smaller tiles, FEM can be used to calculate the strength of electric field under stressing conditions, and the electric field is used to further calculate TDDB Damage and the lifetime of wires and the chip.
- A machine learning based method is proposed to speed up the electrostatics solving process, which aims to solve the electric field in TDDB analysis. After layout partition, each layout tile can be encoded as an image and thus an encoder-decoder network is employed to mimic the FEM solving process. High accuracy along with speedup of two orders are achieved.

- GLU3.0 introduces two main improvements to the previously developed GPU based sparse LU factorization solver GLU. First, a new leveling algorithm is developed, which brings the complexity of the preprocessing stage back to normal. Second, three new modes of GPU kernels are developed, which takes advantage of the data pattern in typical sparse LU factorization and helps better utilize the computing resources of GPU.

1.5 Organization of this thesis

The rest of this thesis is organized as follows. Chapters 2, 3 and 4 introduce the three TDDB analysis works. Chapter 5 introduces GLU3.0. Each chapter begins with an overview, and the related published paper is listed, followed by details of the work and results. Finally, chapter 6 summarizes the thesis.

Chapter 2

Fast TDDB analysis with EPG model

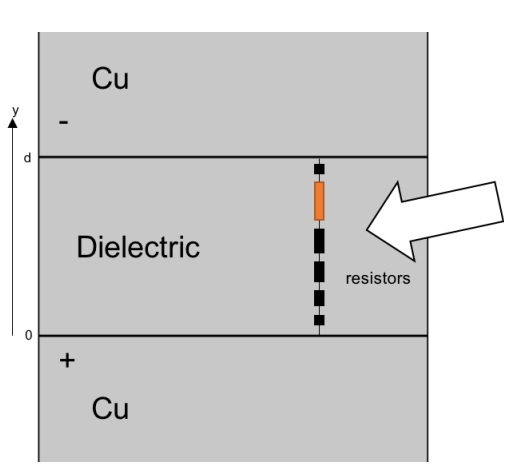
The work presented in this chapter focuses on the EPG model [33], which is based on the breakdown concept of electric path generation. However, determination of the time-to-failure from this model includes time-consuming finite element method (FEM). The work presented in this chapter tries to mitigate this problem by developing fast time to failure (TTF) evaluation method based on the analytic solution of the ion diffusion partial differential equations. It is shown that the location of the minimum concentration can be determined by the dominant terms with sufficient accuracy and the TTF can also be computed with a few dominant terms. On top of this, time-varying stressing voltages are also considered, which are commonly seen in practical VLSI chips. An equivalent DC stressing voltage is computed for given time-varying stressing voltages such that both voltages will lead to the same TTF for the same wire structure. The developed equivalent DC stressing

voltage is parameterized in terms of amplitude, duty cycle, and period for periodic stressing voltage waveforms using regression-based method. The proposed analytic TDDB concentration and TTF formula, and the equivalent DC stressing voltage compact model are all validated against the results of FEM analysis using COMSOL. Numerical results further show the new compact TDDB model can lead to three orders of magnitude speedup with less than 1% error against the existing FEM results. This work is published in [53]

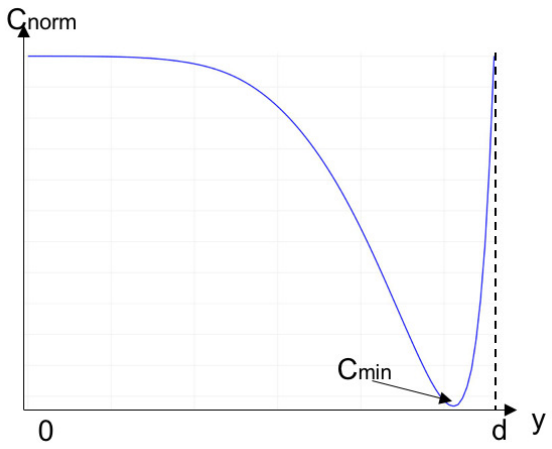
2.1 Review of physics-based TDDB EPG model for low-k BEOL interconnects

The EPG model views dielectric breakdown as a complementary combination of electric current path generation by means of diffusing metal ions and field-based hopping conductivity of the current carriers. Different from the more widely accepted across-layout electrostatic field based TDDB models (i.e. E, $1/E$, and $E^{1/2}$ models), TTF in the EPG model is determined by the kinetics of the electric current path generation, which is controlled by a time-dependent minimum metal ion concentration in IMD. Also, in this model, it is assumed that the barrier metal ions are injected into the dielectrics based on the recent observation [37].

Specifically, 2D diffusion of the metal ions along the cap/IMD interface between the oppositely charged metal wires (as shown in Fig. 1.1a) is considered because experiments demonstrate that TDDB failures take place mostly at this interface [71, 24].



(a) Resistors in IMD along $(0, d)$



(b) Distribution of corresponding ion concentration

Figure 2.1: Dielectric modeled by resistor chain and the corresponding ion concentration

As stated, electric conductivity is represented by electron jumps between neighboring centers (hoping conductivity). The local conductivity is proportional to the probability of the electron jumping between the neighbor centers, which exponentially depends on the distance between the centers [58]:

$$\sigma_{ij} \sim \Gamma_{ij} = \gamma_{ij}^{0} exp\left\{-\frac{2r_{ij}}{a} - \frac{\varepsilon_{ij}}{k_B T}\right\} \tag{2.1}$$

where r_{ij} is the distance between centers marked i and j, a is the radius of electron localization at this type of centers (analog of the Bohr's radius), which can reach $100\mathring{A}$, ε_{ij} is the energy barrier between centers, and $k_B T$ is the thermal energy. All connected centers form a resistor network, with the resistor between i and j centers being

$$R_{ij} = R_{ij}^{0} exp\left\{\frac{2r_{ij}}{a} + \frac{\varepsilon_{ij}}{k_B T}\right\} \tag{2.2}$$

In a 2D system, the distance r_{ij} is determined by: $r_{ij} = C(x, y, t)^{-\frac{1}{2}}$, where $C(x, y, t)$ is the ion concentration at the considered interface. Fig. 2.1 shows the schematic of the distri-

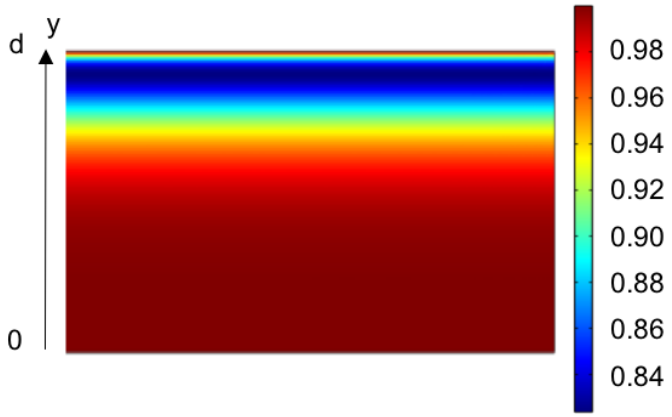


Figure 2.2: Distribution of ion concentrations in one pattern.

bution of ion concentration and corresponding resistor network at an arbitrary instance in time in IMD along a path $(0, d)$ connecting metal electrodes. It is clear that electrons moving from the cathode to anode will meet the biggest resistor at the locations characterized by the largest distance between centers. Since the resistance grows exponentially with the distance, it is reasonable to conclude that the total resistance of the path $(0, d)$ depends on the largest resistor, and its corresponding minimum ion concentration. Fig. 2.2 shows the COMSOL [1] FEM analysis result of the distribution of ion concentration for the pattern in Fig.2.1a under some time of stressing, which is a 2D version of Fig. 2.1b.

The distribution of the normalized ion concentration $C_{\text{norm}}(x, y, t) = C(x, y, t)/C_0$ is governed by the diffusion of ions in an electric field [29]:

$$\frac{\partial C_{\text{norm}}}{\partial t} = -\nabla J, \text{ where } J = -D\nabla C_{\text{norm}} + v_d C_{\text{norm}} \tag{2.3}$$

subject to the following boundary conditions:

$$C_{\text{norm}}(x = 0) = C_{\text{norm}}(x = d) = 1$$

where J is the metal flux, $D = D_0 exp(-Ea/k_B T)$ is the diffusion coefficient, $v_d = E\frac{Dq}{k_B T}$ is the metal ion drifting velocity, Ea is the activation energy for metal ion diffusion, k_B is the Boltzmann constant, T is the temperature, q is the electric charge and E is the electric field.

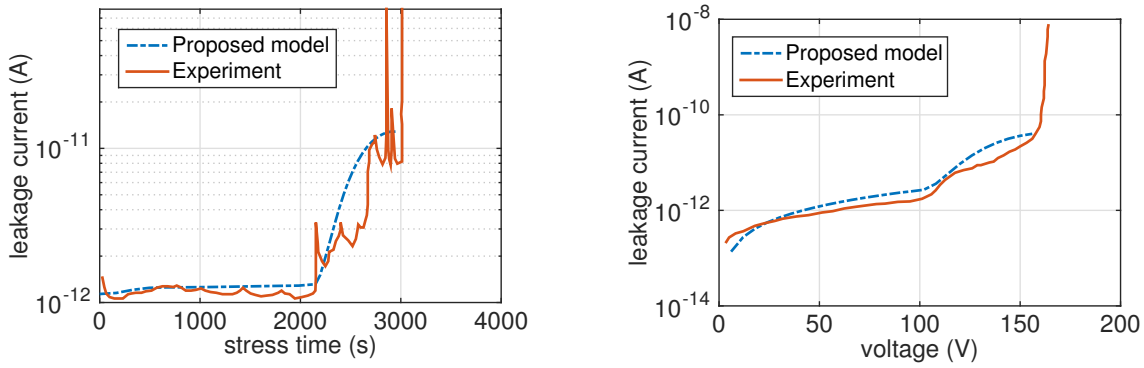
The developed model of the electric path generation and evolution allows derivation of the formalism of the leakage current evolution. As mentioned above, the neighboring ions characterized by the largest separation provides the largest "resistance" for the electrons hopping between metal ions. Assuming that the potential barriers between neighboring centers do not depend on the distance between them ($\varepsilon_{ij} = \varepsilon$), and adopting the Poole-Frenkel mechanism of the field-induced barrier lowering, the expression for the current density evolution can be derived:

$$j(t) = j_0 E exp\left\{-\frac{2}{a\sqrt{C_{\text{norm}}^{\min}(t) \cdot C_0}} - \frac{\varepsilon - q\sqrt{qE/(\pi\varepsilon_{\text{perm}})}}{k_B T}\right\} \tag{2.4}$$

where $\varepsilon_{\text{perm}}$ is the dielectric dynamic permittivity. The total leakage current can be obtained by integral of leakage current density over the whole shape contour. It was shown in [33] that the EPG model agrees with some observed experimental results in terms of breakdown leakage currents over time as shown in Fig. 2.3.

2.2 Faster TDDB analysis flow with the EPG model

This section presents a faster way to analyze ion concentration and TTF in EPG model based on the solution of the ion diffusion equation (2.3). Before proceeding, we show



(a) Constant voltage stress

(b) Ramped voltage stress

Figure 2.3: Current-voltage leakage changes for different voltage (TDDB) stresses modeled by EPG model, compared with experimental data from [61], courtesy of [33].

Figure 2.4: Two example patterns with the same minimum distance (50nm) studied in FEM analysis using COMSOL.

Table 2.1: Parameters and constants used

Paras	**Value**	**Paras**	**Value**
D_0	$2.24 \times 10^{-11}\text{m}^2/\text{s}$	E_a	0.8eV
$\varepsilon_{\text{perm}}$	2.9	k_B	$1.38 \times 10^{-23}\text{J/K}$
T	370K	V_{DD}	1V

Table 2.2: Statistics of TTFs in different patterns with the same minimum distance (50nm)

TTF(s)	**Count**	**Percentage**
360000	42	7.61%
370000	269	48.73%
380000	194	35.14%
390000	34	6.16%
400000	11	1.99%
410000	2	0.04%

Table 2.3: TTF results after changing D_0 or E_a

Test cases	**TTF(s)**
Original case	363426
D_0 increased by 10%	327083
D_0 decreased by 10%	399767
E_a increased by 10%	6113735
E_a decreased by 10%	21604

that the minimum distance between metal wires is the dominant factor in determining the TTF as this is the starting point of this work.

FEM analysis of equation (2.3) using COMSOL has been done to simulate TTFs of 552 different patterns. In all these patterns, there are two metal wires and the minimum distance between them are the same. Two example patterns are shown in Fig. 2.4, where the two gray lines represent metal wires, one connected to VDD (supply voltage), and the other connected to GND (ground), and the white area is the dielectric in between. $C_x = 0.95$ is employed as the threshold normalized minimum concentration leading to failure. The parameters and constants used in the FEM analysis are shown in Table 2.1. The diffusion coefficient D_0 and activation energy E_a are obtained from fitting the experimental results in [61] as shown in Fig. 2.3.

Table 2.4: Statistics of TTFs in different patterns with the same minimum distance (20nm)

TTF(s)	**Count**	**Percentage**
58000	23	4.17%
59000	159	28.80%
60000	219	39.67%
61000	78	14.13%
62000	30	5.43%
63000	12	2.17%
64000	18	3.26%
65000	13	2.36%

First, a test is done with patterns with minimum distance of 50nm. The results are listed in Table 2.2, where *Count* means the number of patterns with the corresponding TTF value. It is shown that the TTFs for most cases are in the range between 360000s and 41000s for the minimum spacing of 50nm. Note that D_0 and E_a have huge impacts on the TTFs of TDDB as they determine the diffusion speeds of the barrier metal ions. We changed the two parameters and the results are shown in Table 2.3.

Table 2.4 shows the TTF range is between 58000s and 65000s for a second test with the minimum spacing of 20nm. The precision of this range is limited by the timestep set in COMSOL. It shows that the layer pattern has some impact on the TTF values, and this impact gets more significant for smaller minimum spacing. However, for both 50nm and 20nm minimum spacings, the impact on TTF is quite small for the majority of the patterns. It is also observed that the minimum spacing dominant effect becomes more obvious when the length of the wires is sufficiently long (compared to the spacing between them). Based on this observation, as a first-order approximation, the original problem can still be simplified to the one-dimensional problem with two parallel metal wires separated

by the minimum distance like the one shown in Fig. 2.1a, which is the starting point of this work.

2.2.1 Analytic solution of ion concentration in the IMD

The normalized concentration $C_{norm}(x,t)$ is simplified as $C(x,t)$ for the sake of better presentation. Then equation (2.3) can be rewritten as

$$D\nabla^2 C = \frac{qD}{k_B T}\nabla(C \cdot E) + \frac{\partial C}{\partial t} \tag{2.5}$$

with boundary condition:

$$C(\text{at the edge of dielectric}) = 1 \tag{2.6}$$

and initial condition:

$$C(t = 0, \text{ within the dielectric}) = 0 \tag{2.7}$$

Equation (2.5) can be further simplified to one-dimensional form given the previous discussion as follows:

$$D\frac{\partial^2 C}{\partial x^2} = \frac{qDE}{k_B T} \cdot \frac{\partial C}{\partial x} + \frac{\partial C}{\partial t} \tag{2.8}$$

with boundary condition:

$$C(x = 0) = C(x = L) = 1 \tag{2.9}$$

and initial condition:

$$C(x,t) = 0, 0 < x < L, t = 0 \tag{2.10}$$

Further define X, T', λ, and L as

$$X = \frac{x}{L}, \quad T' = \frac{Dt}{L^2}, \quad \lambda = \frac{P_e}{2} = \frac{qE}{2k_B T}L = \frac{qV_{DD}}{2k_B T} \tag{2.11}$$

Equation (2.8) can be written as

$$\frac{\partial^2 C}{\partial X^2} = P_e \frac{\partial C}{\partial X} + \frac{\partial C}{\partial T'} \tag{2.12}$$

The solution of (2.12) can be found by using Laplace transformation based method as shown below. The detailed derivation of the solution can be found at the Appendix section.

$$C(X, T') = 1 - 2\pi \sum_{n=1}^{\infty} \frac{n \sin(n\pi X)}{(n^2\pi^2 + \lambda^2)} e^{-(n^2\pi^2+\lambda^2)T'} \cdot \left[e^{\lambda X} - (-1)^n e^{\lambda X - \lambda} \right] \tag{2.13}$$

Note that equation (2.13) is an exact analytic solution without any approximation.

2.2.2 TDDB time to failure estimation

This section introduces three different methods to estimate TTF based on different approximation of equation (2.13). An easy yet inaccurate method is first introduced, followed by an accurate but inefficient version. Finally, a combined method of these two that is both accurate enough and efficient is introduced.

Before proceeding, it is worthwhile to note that the goal here is to find TTF, which is the time when $min\,[C(X, TTF)]$ reaches the threshold normalized minimum concentration C_X for $0 < X < 1$.

As equation (2.13) contains an infinite series, it is not possible to get an analytical result of TTF. However, numerical methods are still feasible. First, it is natural to consider using first term approximation because of its simplicity. The first term approximation of equation (2.13) is:

$$C_1(X, T') = 1 - \frac{2\pi \sin(\pi X)}{\lambda^2 + \pi^2} e^{-(\lambda^2+\pi^2)T'} \left(e^{\lambda X} + e^{\lambda X - \lambda} \right) \tag{2.14}$$

This equation can be used to derive where the minimum concentration locates (labeled by X_{fail}) by calculating the partial derivative of concentration with respect to X and solve for the zero points. Then, a fixed X_{fail} which does not change with T' can be derived:

$$X_{fail} = 1 - \frac{1}{\pi} \arctan\left(\frac{\pi}{\lambda}\right) \tag{2.15}$$

This X_{fail} can be used back in equation (2.14) to solve for TTF, which gives the result:

$$TTF = \frac{L^2}{D(\lambda^2 + \pi^2)} \ln \frac{2\pi^2 \left(e^{\lambda - (\lambda/\pi)\arctan(\pi/\lambda)} + e^{-(\lambda/\pi)\arctan(\pi/\lambda)}\right)}{(1 - C_X)\left(\lambda^2 + \pi^2\right)^{3/2}} \tag{2.16}$$

The details of this derivation can be found at the Appendix section. This is the first method to evaluate TTF. However, the results from above equation (2.16) are not accurate as shown in Section 2.4.

The second method which is quite accurate contains two iterative phases, namely X and T' phases. First, in order to conduct sufficiently accurate approximation of equation (2.13), it is preferred to use a large number of terms (10000 terms are used in experiments) of the infinite series. Then, because the concentration is monotonically increasing with respect to T', so the bisection method can be used to numerically find TTF efficiently on condition that X is fixed. This is the T' phase, which is named because it solves T' with X fixed. Issues about X_{fail} can be tackled by the fact that there exists only one minimum value of concentration with respect to X. Therefore in X phase X_{fail} can be searched step by step with a fixed T' from the previous T' phase. In detail, X is adjusted with a small enough X_{step} in each searching step until the minimum concentration at X_{fail} is found. Then, a new T' phase may be started for more accurate TTF. The final result can be found by performing two phases iteratively for only a few times.

Table 2.5: Fitted coefficients of the power law model

Temperature(K)	$B(T)$	N
370	exp(12.839)	-0.928
390	exp(11.612)	-0.914
410	exp(10.508)	-0.9

This is the most accurate yet most complicated method. In fact, the previous two methods can be combined together to get an accurate and simple method.

As discussed above, the bisection method can be used to numerically find TTF if we use 3 or 5 terms for approximation. Note that an odd number of terms is used because the function of concentration has to be monotonic. At the same time, X_{fail} given by equation (2.15) can be directly used so the X phase can be skipped. The latter Section 2.4 shows that results derived from this simple method are quite accurate. In short, the X_min in equation (2.15) can be used and then the 3 terms approximation based bisection method can be employed to numerically derive an accurate enough TTF.

A pseudocode describing this evaluation flow through bisection method is listed in Algorithm 1. $C_{min}(T')$ stands for the ion concentration calculated through a certain approximation of equation (2.13) (for example, 3-terms approximation), with $X = X_{fail}$ from equation (2.15).

2.2.3 Study of the relationship between TTF and electric field

This section focuses on the relationship between the TTF and stressing electric filed or voltage for the EPG TDDB model. It is shown that EPG model shows similar

Algorithm 1 Evaluation of TTF

1: Set parameters such as spacing L, stressing voltage V_{DD}

2: Calculate X_{fail} from equation (2.15)

3: Set initial $T' = 1$, result error tolerance *eps*

4: **while** $C_{min}(T') < C_X$ **do**

5: $T' = T' * 2$

6: **end while**

7: $a = 0, b = T', c = (a + b)/2$

8: **while** $b - a < eps$ **do**

9: **if** $(C_{min}(a) - C_X) * (C_{min}(c) - C_X) < 0$ **then**

10: $b = c, c = (a + b)/2$

11: **else if** $(C_{min}(b) - C_X) * (C_{min}(c) - C_X) < 0$ **then**

12: $a = c, c = (a + b)/2$

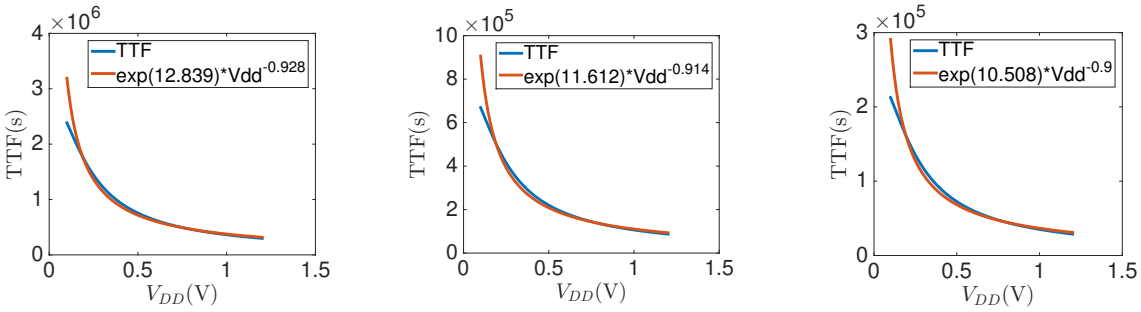
13: **else**

14: break

15: **end if**

16: **end while**

17: $T'_{fail} = c$



(a) 370K (b) 390K (c) 410K

Figure 2.5: TTF against different stressing voltages (V_{DD}) under different temperatures.

relationship between TTF and electric field of the power law model [43, 68]:

$$TTF = B(T)V^N \tag{2.17}$$

To illustrate this, TTF is calculated with the most accurate method introduced in Section 2.2.2 over different electric fields. The pattern studied is still the same pattern used above. Only supply voltage V_{DD} is sweeped so that electric field can be changed. Fitted coefficients of the power-law model under different temperatures are shown in Table 2.5. Furthermore, Fig. 2.5 show the TTF predicted by the fitting power law models against our EPG models under different temperatures.

The fitted voltage exponent N is around 0.9 for different temperature, which is quite smaller than other power law models with N ranging from 40 to 48 [43] or even 24-36 [68]. In addition, the recent experimental data from the published BEOL TDDB power law models show that the exponent N is around 20-24 [16, 46]. However, in the EPG model, the material (Ta) and structure (confined copper with Ta as the barrier layer and low-k dielectrics) are totally different with those experiment settings. The underlying

breakdown physics is also different. As a result, the fitted exponent, N, can be significant different than the existing power law models.

2.3 Equivalent DC stressing voltage analysis of EPG model

In the previous discussion, it is assumed that two metal wires in the pattern are connected to VDD and GND respectively. In real digital ICs, this scenario does not apply to every two neighboring wires, as there are other kinds of wires such as clock net and signal net. However, it is still feasible to assume that the voltages on both wires can be modeled as DC or a square wave. With this being said, it is obvious that the electric field within the IMD can be modeled as a square wave. For simplicity, it is assumed to be unipolar and it covers the cases where voltage of one metal wire is constant. Three parameters can be used to define this unipolar square wave: amplitude, period, and duty cycle. With them, a function can be fitted to convert any unipolar square wave to the equivalent DC stressing voltage provided that they result in the same TTF. Other cases resulting in bipolar square wave can be treated with the same method, but with one more parameter.

Before moving on, we first review the response surface method (RSM), which is used in this study. RSM, consisting of a group of mathematical and statistical techniques, explores the relationships between several input variables and one or more responses. Based on a set of designed experiments, an optimal response can be obtained [48]. Specifically in RSM, input parameters are called independent variables and performance measure is considered as a response. Response y depends on input independent variables $(\xi_1, \xi_2, \ldots, \xi_k)$

$$y = f(\xi_1, \xi_2, \ldots, \xi_k) + \varepsilon \tag{2.18}$$

where f is the true response function which is unknown and could be very complicated and ε is an error of the model. Usually, a low-order polynomial in some relatively small region of the independent variable space is appropriate. First order model or second order model are the most commonly used models in RSM. In this article, second order model with relatively good accuracy is employed. A second-order response y depending on variables $(x_1, x_2, \ldots, x_k)$ can be written as:

$$y = \beta_0 + \sum_{j=1}^{k} \beta_j x_j + \sum_{j=1}^{k} \sum_{i \leq j} \beta_{ij} x_i x_j + \varepsilon \tag{2.19}$$

Let $x_{k+1} = x_1 x_1, x_{k+2} = x_1 x_2, \ldots, x_{k(k+3)/2-1} = x_{k-1} x_k, x_{k(k+3)/2} = x_k x_k$ and $\beta_{k+1} = \beta_{11}, \beta_{k+2} = \beta_{12}, \ldots$ then equation (2.19) can be expressed as

$$y = \beta_0 + \sum_{j=1}^{q} \beta_j x_j + \varepsilon \tag{2.20}$$

which is a linear regression model for coefficients $(\beta_0, \beta_1, \ldots, \beta_q)$, where $q = k(k+3)$. With equation (2.20), least square method is employed to estimate the regression coefficients in the multiple linear regression model.

We assume that there are n observed responses $y = (y_1, y_2, \ldots, y_n)$ and for each y_i, there are corresponding parameters $x_i = (x_{i1}, x_{i2}, \ldots x_{iq})$. So equation (2.20) can be written in matrix notation as follow:

$$y = X\beta + \varepsilon \tag{2.21}$$

To solve β in a least square minimization sense, QR decomposition on X is applied. It is shown that $R\beta = Q^T y$. After solving the linear equations, estimated coefficient vector $\hat{\beta}$ can be calculated. In this work, the input variables x_i are normalized parameters, which include amplitude, period, and duty cycle.

The TTFs for DC stressing voltage are calculated using methods introduced in Section 2.2.2. A formula has been fitted to quickly find the equivalent DC voltage with the given TTF. TTFs for different square wave stressing voltage are derived using COMSOL.

Because of the limitation of timestep in FEM analysis, the period of square wave must be very long compared to practical cases.

2.4 Experimental results and discussions

This section shows the numerical results and comparisons for the new TDDB analysis flow. All programs run on a workstation with Xeon E5-2698 CPU and 128GB memory.

The first experiment is to test the accuracy of the derived analytic solution in equation (2.13) by comparing it with the results from COMSOL FEM analysis. The results are shown in Table 2.6. The data is based on the pattern shown in Fig. 2.1a, in which the distance between two metal wires is 50nm, and temperature is 370K. The metal wire at $x = 0$ is connected to VDD and the wire at $x = 50$ is connected to GND. As shown in the table, the proposed closed form expression give very accurate results compared to COMSOL simulations.

The second experiment is about the accuracy and calculation CPU time of different methods for TTF based on the same pattern above. $C_x = 0.95$ as the threshold normalized minimum concentration leading to failure in the experiment. For the comparison, the result using 10000 terms and the resulting changes in X_{fail} is used as the golden results. The comparison results are shown in Table 2.7. In the last four rows, X_{fail} from equation (2.15)

Table 2.6: Results of ion concentration derived from two methods

Time(s)	**x(nm)**	**FEM**	**Results from** (2.13)	**Error (%)**
1×10^5	35.97	0.01377	0.01363	1.03%
1×10^5	46.56	0.11680	0.11485	1.68%
2×10^5	11.67	0.99512	0.99527	0.01%
2×10^5	33.62	0.66238	0.66012	0.34%
3×10^5	20.49	0.99691	0.99754	0.06%
3×10^5	42.83	0.85763	0.85553	0.24%

Table 2.7: TTF from different methods, the calculation times and speedup over FEM

Method	**TTF(s)**	**Error (%)**	**Time(s)**	**Speed-up**
10000 terms with exact X_{fail}	363426	-	8.702	4.94
FEM	370000	1.81%	43.000	–
10000 terms	363221	0.06%	3.468	12.40
5 terms	363221	0.06%	0.013	$3.31E3$
3 terms	366562	0.86%	0.009	$4.78E3$
1 term	412568	13.52%	0.004	$1.08E4$

is used instead of more accurate X_{fail} for the sake of simplicity. Column **Error(%)** is the relative error compared to the golden. **Times(s)** is the CPU times and column **Speed-up** is the speedup against FEM.

The table shows that if 5 terms are used, the result is sufficiently accurate compared to 10000 terms as they can give almost the same results. If only 3 terms are used, the results can be very accurate as well. In terms of CPU time and speedup, the speedup ranges from 15 to 1.08×10^4. Results from 3-terms show a good compromise between accuracy and speed, which gives about 0.86% accuracy, and the speedup more than three order of magnitude (4.78×10^3).

The same experiment with the spacing of $20nm$ is also done using 3-terms method. The result of TTF is $58650s$. Basically, the analytic solution (2.13) shows that if only spacing changes, then it holds that $TTF \propto L^{-2}$.

Last but not least, the RSM fitted results are tested by calculating the equivalent DC stressing voltage for different stressing voltages and comparing the results of TTF from COMSOL. The fitted function to calculate equivalent voltage when $T = 370$K is:

$$\begin{aligned} V_{eq} =& 0.4794 + 0.1606X_1 + 0.4675X_2 + 0.0185X_3 \\ &+ 0.0075X_1^2 + 0.1563X_1X_2 + 0.0133X_1X_3 \\ &- 0.0323X^2 - 0.0252X_2X_3 - 0.0031X_3^2 \end{aligned} \tag{2.22}$$

where

$$X_1 = \frac{\text{Amplitude(V)} - 0.9}{0.3} \quad X_2 = \frac{\text{Duty cycle} - 0.54}{0.46} \quad X_3 = \frac{\text{Period(s)}/1000 - 120}{80}$$

Some tests with parameters given in Table 2.8 have been done to test the correctness of this function. All these test cases are not used in building the RMS function. Each time only one of the three parameters is changed so a graph showing the impact of this variable can be drawn. The graphs are shown in Fig. 2.6. In the graphs, the COMSOL results are derived by searching for equivalent DC voltages based on TTF results from COMSOL. It can be seen that these graphs prove that RMS gives a good result for evaluating equivalent DC voltage.

It is worth mentioning that in the cases with the changeable distance between two metal wires, this result can still be used in that the distance L does not appear in equation (2.13). The only thing it can affect is the relationship between T' and t, as is

Table 2.8: Parameters for 3 tests

Amplitude(V)	**Duty cycle**	**Period(s)**
0.9	0.25∼0.85	100000
0.9	0.45	25000∼175000
0.5∼1.3	0.65	140000

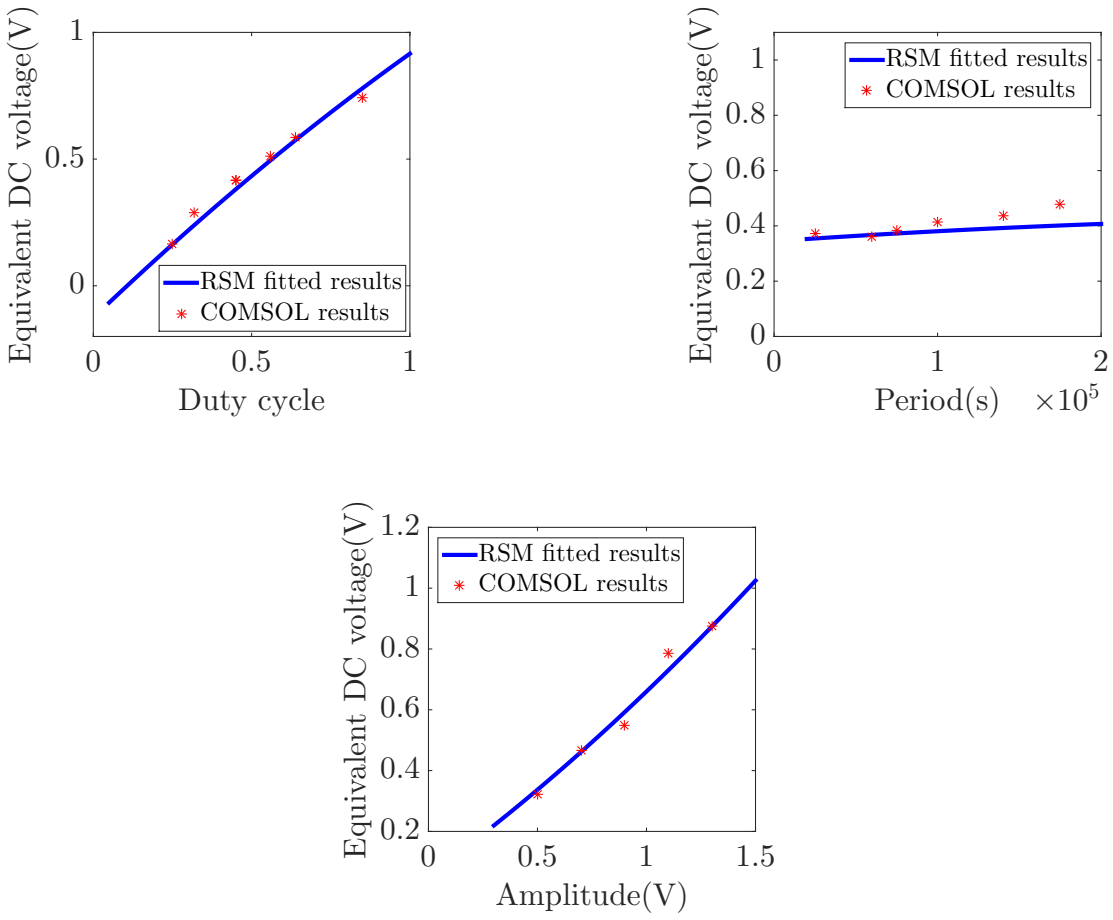


Figure 2.6: Comparison between RSM results with COMSOL results on different parameters.

introduced in equation (2.11). So distance does not change the function of equivalent DC voltage at all.

2.5 Summary

The work presented in this chapter introduces a fast way to evaluate TDDB life time based on the EPG TDDB model. It is based on the analytic solution of the ion diffusion partial differential equation. It is shown that the location of the minimum concentration can be determined by the dominant terms and the TTF can be computed by using a few dominant terms. On top of this, the method is extended to cover time-varying stressing voltage, which is commonly seen in practical VLSI chips. The equivalent DC stressing voltage, parameterized by amplitude, duty cycle, and period for periodic stressing voltage waveforms, is calculated using regression based method. Numerical experiments validate that the proposed analytic TDDB concentration and TTF formula and the equivalent DC stressing voltage compact model against the results from FEM analysis using COMSOL. It is further shown that the new compact TDDB model can lead to three orders of magnitude speedup with less than 1% error.

2.6 Appendix

2.6.1 Derivation of the analytic solution for the ion diffusion equation

In this section, we give the detailed mathematical derivation of the formulas for the proposed TDDB compact model. For the completeness of this section, the metal ion

diffusion equation (2.5) is rewritten as follows:

$$D\nabla^2 C = \frac{qDE}{k_B T}\nabla C + \frac{\partial C}{\partial t} \tag{2.23}$$

with the boundary conditions:

$$C(\text{at the edge of dielectric}) = 1 \tag{2.24}$$

In the one dimensional case, (2.23) can be written as

$$D\frac{\partial^2 C}{\partial x^2} = \frac{qDE}{k_B T}\cdot\frac{\partial C}{\partial x} + \frac{\partial C}{\partial t} \tag{2.25}$$

subject to boundary conditions:

$$C(x=0) = C(x=L) = 1 \tag{2.26}$$

$$C(x,t) = 0, 0 < x < L, t = 0 \tag{2.27}$$

Define:

$$X = \frac{x}{L} \qquad T' = \frac{Dt}{L^2} \qquad P_e = \frac{qE}{k_B T}L = \frac{qV_{dd}}{k_B T} \tag{2.28}$$

It gives:

$$\frac{\partial^2 C}{\partial X^2} = P_e\frac{\partial C}{\partial X} + \frac{\partial C}{\partial T'} \tag{2.29}$$

After the Laplace transformation, it gives

$$\frac{d^2\hat{C}}{dX^2} = P_e\frac{d\hat{C}}{dX} + p\hat{C} \tag{2.30}$$

subject to boundary conditions:

$$\hat{C}(0,p) = \hat{C}(1,p) = \frac{1}{p} \tag{2.31}$$

where p is the Laplace domain variable. For the differential equation (2.30) , its solution would appear in this following general form:

$$\hat{C} = r_1 e^{p_1 X} + r_2 e^{p_2 X} \tag{2.32}$$

where p_1 and p_2 are the roots for equation $x^2 = P_e x + p$. As a result, it gives $p_{1,2} = \frac{P_e}{2} \pm a$, $a = \sqrt{\frac{P_e^2}{4} + p}$ with the following boundary conditions

$$\begin{cases} r_1 + r_2 = \frac{1}{p} \\ r_1 e^{a + P_e/2} + r_2 e^{-a + P_e/2} = \frac{1}{p} \end{cases} \tag{2.33}$$

By solving for r_1 and r_2, one can obtain the solution in Laplace domain as

$$\hat{C}(p) = \frac{e^{P_e X/2} \sinh\left[a(1-X)\right]}{p \sinh(a)} + \frac{e^{P_e (X-1)/2} \sinh(aX)}{p \sinh(a)} \tag{2.34}$$

Inverse Laplace transform on (2.34) is then done to obtain the solution in the time domain. First take a look at the first part of (2.34), which is

$$\hat{C}_1 = \frac{e^{P_e X/2} \sinh\left[a(1-X)\right]}{p \sinh(a)}$$

A property of Laplace transform as this is used here:

$$\mathcal{L}^{-1}\left\{\hat{C}(X, p+b)\right\} = e^{-bT'} \mathcal{L}^{-1}\left\{\hat{C}(X, p)\right\}$$

Then it gives

$$C_1 = e^{\lambda X - bT'} \cdot \mathcal{L}^{-1}\left\{\frac{\sinh\left[p^{1/2}(1-X)\right]}{(p-b)\sinh(p^{1/2})}\right\} \tag{2.35}$$

where $b = \frac{P_e^2}{4}$. Assume $\lambda = b^{1/2} = \frac{P_e}{2}$. The inverse transform can be derived by adding residues at all poles, in this case there is a pole at $p = b$, and poles at $p^{1/2} = n\pi i, n =$

$0, \pm 1, \pm 2 \ldots$, or equally $p = -n^2\pi^2, n = 0, 1, 2 \ldots$

$$\begin{aligned}
\mathcal{L}^{-1}\left\{\frac{\sinh\left[p^{1/2}(1-X)\right]}{(p-b)\sinh(p^{1/2})}\right\} &= \int_C \frac{\sinh\left[p^{1/2}(1-X)\right]}{(p-b)\sinh(p^{1/2})} e^{pT'} dp \\
&= \frac{\sinh\left[\lambda(1-X)\right]}{\sinh(\lambda)} e^{\lambda^2 T'} - 2\pi \sum_{n=1}^{\infty} \frac{n\sin(n\pi X)}{(n^2\pi^2+\lambda^2)} e^{-n^2\pi^2 T'}
\end{aligned} \tag{2.36}$$

The second part can be treated in the same way. After the inverse transformation, two parts are combined. With some additional algebraic operations the final result is obtained:

$$\begin{aligned}
& C(X,T') \\
&= e^{\lambda X}\left\{\frac{\sinh\left[\lambda(1-X)\right]}{\sinh(\lambda)} - 2\pi\sum_{n=1}^{\infty}\frac{n\sin(n\pi X)}{(n^2\pi^2+\lambda^2)} e^{-(n^2\pi^2+\lambda^2)T'}\right\} \\
&\quad + e^{\lambda X-\lambda}\left\{\frac{\sinh(\lambda X)}{\sinh(\lambda)} + 2\pi\sum_{n=1}^{\infty}\frac{(-1)^n n\sin(n\pi X)}{(n^2\pi^2+\lambda^2)} e^{-(n^2\pi^2+\lambda^2)T'}\right\} \\
&= \frac{e^{\lambda X}\sinh\left[\lambda(1-X)\right] + e^{\lambda X-\lambda}\sinh(\lambda X)}{\sinh(\lambda)} \\
&\quad - e^{\lambda X} 2\pi\sum_{n=1}^{\infty}\frac{n\sin(n\pi X)}{(n^2\pi^2+\lambda^2)} e^{-(n^2\pi^2+\lambda^2)T'} \\
&\quad + e^{\lambda X-\lambda} 2\pi\sum_{n=1}^{\infty}\frac{(-1)^n n\sin(n\pi X)}{(n^2\pi^2+\lambda^2)} e^{-(n^2\pi^2+\lambda^2)T'} \\
&= 1 - 2\pi\sum_{n=1}^{\infty}\frac{n\sin(n\pi X)}{(n^2\pi^2+\lambda^2)} e^{-(n^2\pi^2+\lambda^2)T'}\left[e^{\lambda X} - (-1)^n e^{\lambda X-\lambda}\right]
\end{aligned} \tag{2.37}$$

2.6.2 Derivation of analytic form for time to failure

As approximation, let's take the first term of the analytic solution $C(X,T')$ as $C_1(X,T')$:

$$C_1(X,T') = 1 - \frac{2\pi\sin(\pi X)}{\lambda^2+\pi^2} e^{-(\lambda^2+\pi^2)T'}\left(e^{\lambda X} + e^{\lambda X-\lambda}\right) \tag{2.38}$$

To get the minimum concentration, the derivative of $C_1(X, T')$ is set to zero:

$$\begin{aligned}&\frac{\partial C_1\left(X, T'\right)}{\partial X}=0\\&=\frac{2\pi e^{-\left(\lambda^2+\pi^2\right)T'}}{\lambda^2+\pi^2}e^{\lambda X}\left(1+e^{-\lambda}\right)\left[\lambda\sin\left(\pi X\right)+\pi\cos\left(\pi X\right)\right]\end{aligned}\tag{2.39}$$

The solution of this equation gives a fixed X_{fail} which does not change with T' as

$$X_{fail}=1-\frac{1}{\pi}\arctan\left(\frac{\pi}{\lambda}\right)\tag{2.40}$$

Then at X_{fail}, define $C\left(X_{fail}, T'\right)=C_{fail}$ as the minimum concentration that triggers the dielectric failure

$$\begin{aligned}C_{fail}=1-&\frac{2\pi e^{-\left(\lambda^2+\pi^2\right)T'}}{\lambda^2+\pi^2}\sin\left[\pi-\arctan\left(\pi/\lambda\right)\right]\cdot\\&\left(e^{\lambda-(\lambda/\pi)\arctan(\pi/\lambda)}+e^{-(\lambda/\pi)\arctan(\pi/\lambda)}\right)\end{aligned}\tag{2.41}$$

$$\begin{aligned}=1-&\frac{2\pi^2 e^{-\left(\lambda^2+\pi^2\right)T'}}{\left(\lambda^2+\pi^2\right)^{3/2}}\cdot\\&\left(e^{\lambda-(\lambda/\pi)\arctan(\pi/\lambda)}+e^{-(\lambda/\pi)\arctan(\pi/\lambda)}\right)\end{aligned}\tag{2.42}$$

The resulting time to failure can be found by solving:

$$C_{fail}=C_X$$

$$T'_{fail}=\frac{1}{\lambda^2+\pi^2}\ln\frac{2\pi^2\left(e^{\lambda-(\lambda/\pi)\arctan(\pi/\lambda)}+e^{-(\lambda/\pi)\arctan(\pi/\lambda)}\right)}{\left(1-C_X\right)\left(\lambda^2+\pi^2\right)^{3/2}}\tag{2.43}$$

$$t_{fail}=\frac{L^2}{D(\lambda^2+\pi^2)}\ln\frac{2\pi^2\left(e^{\lambda-(\lambda/\pi)\arctan(\pi/\lambda)}+e^{-(\lambda/\pi)\arctan(\pi/\lambda)}\right)}{\left(1-C_X\right)\left(\lambda^2+\pi^2\right)^{3/2}}\tag{2.44}$$

Chapter 3

Full-Chip Wire-Oriented TDDB Analysis

The simplified TDDB evaluation method introduced in chapter 2 focuses on the spacing of IMD. For simplicity, the method treats all interconnect structures with same distance as parallel structures. This, however, ignores the complicated geometries of all the layout patterns, which is related to TDDB lifetime. Also, the results of TTF in fact represent the probability of failure. The breakdown can happen anywhere in the IMD analyzed. As a result, for more practical and accurate results, the overall probability of failure of the given IMD should take in consideration the length and shape of it [3].

Based on this idea, the work presented in chapter proposes a full-chip TDDB failure analysis methodology to evaluate lifetime and identify TDDB hotspots in VLSI layouts, which are essentially interconnect wires that have high failure risk due to TDDB. The proposed method features three new techniques compared to existing methods. First,

a partition based scheme is developed to deal with the vast scale of full-chip analysis by partitioning the full chip layout into smaller tiles. Second, for each tile, a newly-introduced TDDB failure metric called *TDDB Damage* is calculated for all wires. Such a wire-oriented TDDB analysis is the first of its kind and is very amenable for physical design as the wires can be easily adjusted or re-routed for TDDB-aware optimization. Third, the new method considers the impact of the non-uniform electric field calculated using the finite element method (FEM), which significantly improves the accuracy of TDDB risk evaluation. Experimental results show that the proposed new TDDB analysis method is more accurate than a recently proposed full-chip TDDB analysis method in which electrical field is treated as a constant value. Additionally, the proposed method can analyze a practical VLSI layout in a few hours. This work is published in [50].

3.1 Overview

As stated, simple interconnect leakage test structures such as serpentine-comb or comb-comb have been typically used in the studies of BEOL TDDB [67]. In contrast, very little work has been done to develop a framework for assessing BEOL TDDB risk in practical and actual circuit layout geometries, which this work focuses on. The difference between the two are two folds. First, the probability of failure of IMD grows with its length. Second, complicated layout geometries can also play an important role.

The effect of layout geometries can be seen in a simple simulation. The results through FEM analysis of electric field in a small piece of a synthesized layout shown in Fig. 3.1 reveals that with a voltage difference between neighboring wires, the electric field

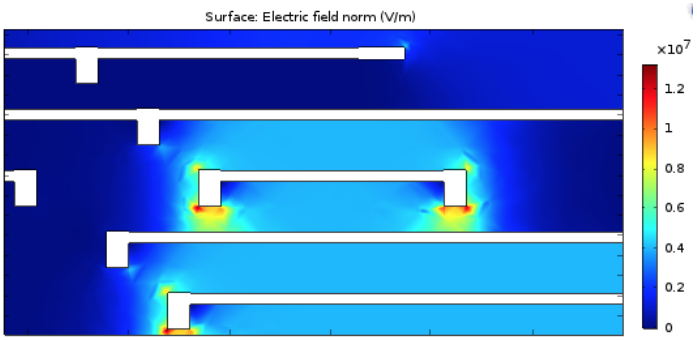


Figure 3.1: FEM analysis result showing higher electric field around tips

at "corners" or "tips" of wires can actually differ greatly from that along longer, straight-line segments, leading to the so called "layout effects' [32]. The higher electric field at tips can substantially reduce the lifetime of such wires, making the results scaled from test data optimistic and inaccurate. Thus, more applicable methods considering such layout effects are required if experimental data is to be used in practical lifetime analysis.

Because this method is based on solving electric field, unlike [3], it covers the non-uniform electrical field and cover local field enhancement effects, which leads to more accurate TDDB lifetime estimation. Additionally, the proposed methods can be applied to all the existing electric field-dependent TDDB models such as E, 1/E, power-law, and $\sqrt{E}$ models. Numerical results show that the proposed method can identify the TDDB hotspots for a practical VLSI layout in a reasonable amount of time. It is further shown that M1 (Metal1) typically has the shortest lifetime because of the highest metal density.

There are three recent publications focusing on full-chip BEOL TDDB assessment. A chip-scale TDDB lifetime simulator is proposed in [3]. Using data from tests with same wire space makes the evaluation process straightforward, which however also means that the layout effect is not considered, making the results scaled from test data optimistic and

inaccurate. Thus, it is not sufficient to evaluate lifetime purely by extracting spacings and scaling from test data assuming a uniform electrical field. Another limitation of [3] is that all horizontal wire segments and vertical ones are considered separately. In reality, one particular interconnect wire can have horizontal and vertical parts at the same time. Hence, they should not be analyzed separately.

The same problem of layout effect also applies to the work presented in chapter 2 [53]. It is based on different TDDB equations, so accuracy is compromised when the assessment is simplified to focus on spacing between wires alone.

The simulator introduced by [33] is built on a look-up table of high-risk interconnect shape patterns. Although local field enhancement is considered in preparation of the look-up table, the effect of length scaling of wires is not covered.

3.2 Length-aware TDDB model

This section first briefly reviews several TDDB models, especially the $\sqrt{E}$ model that is used in this work, then the equations are derived to evaluate TDDB lifetime and failure rate based on the $\sqrt{E}$ model. Note that although $\sqrt{E}$ model is used throughout this study, any other electric field dependent model can be used in the same way with the proposed analysis flow.

3.2.1 Review of $\sqrt{E}$ and other TDDB models

Various models have been proposed based on different assumptions of the physics of BEOL TDDB. Some commonly discussed models are $\sqrt{E}$, power-law, E, $1/E$, and impact

damage model [67]. All of them can fit the measured data obtained at high-E field stressing conditions well. However, they differ substantially when extrapolated to the practical low-E use conditions. As some further experimental results have shown [40], $\sqrt{E}$ model performs relatively better in fitting measured results under both high-E field and low-E field stressing conditions in conventional dielectrics. Thus, $\sqrt{E}$ model is used in this work. However, the specific choice of model does not limit the applicability of the this work, as all models are based on the electric field, and electric field is solved in the proposed analysis flow.

The $\sqrt{E}$ model indicates that TDDB lifetime can be described in the following general form [18, 6]:

$$TTF = K \exp\left(-\gamma\sqrt{E}\right) \exp\left(\frac{E_a}{kT}\right) \tag{3.1}$$

where TTF is the time to failure or lifetime, K is a fitting constant, γ is the electric field acceleration factor, E represents the electric field, E_a is the Arrhenius activation energy, k is the Boltzmann constant, and T is the absolute temperature. The $\sqrt{E}$ dependence comes from the assumption that Poole-Frenkel [60] or Schottky Emission injection [6] plays the primary role in electron conduction, and they both have a $\sqrt{E}$ dependence.

3.2.2 TDDB damage model of interconnect wires

In studies validating the $\sqrt{E}$ model [5], coefficients in (3.1) were fitted based on measurements carried out on a large set of identical interconnect test structures and geometries, under different stressing conditions.

However, for practical VLSI layouts, the study shows that the TDDB failure is also wire length-dependent [3]. Besides length, there exist significant differences in the shape of

interconnect wires in VLSI layouts, and thus the wire length and shape should be accounted for when the equation is used to estimate TDDB lifetime of wires in practical VLSI layouts. In [69], the reciprocal of the Weibull shape factor β is used as the area scaling factor for gate oxides. Thus, time to failure is proportional to $A^{-\frac{1}{\beta}}$ for the same stressing conditions, where A is the area of the dielectric. Based on this observation, in this work β is used as the *length* scaling factor for interconnect wires. This scaling approach has also been used in other BEOL TDDB lifetime analysis publications [3]. Note that scaling is applied to perimeter length instead of area because it is safe to assume that the interconnect thickness is relatively constant, and thus the area scaling term reduces to a length scaling term with an appropriate adjustment to the fitting constant. As a result, in this work, the following TDDB lifetime formula considering the length effect or scaling is used, which is shown below [69]:

$$TTF = K' \exp\left(-\gamma\sqrt{E}\right) \exp\left(\frac{E_a}{kT}\right) L^{-\frac{1}{\beta}} \tag{3.2}$$

where K' is a new fitting constant.

A new metric called *TDDB Damage* (D) of wires is further defined here on top of (3.2) by multiplying the reciprocal of TTF, which is used as a reference of lifetime consumed. It is defined as an accumulative factor with a derivative:

$$dD = \frac{dt}{TTF} = \frac{1}{K'} \exp\left(\gamma\sqrt{E}\right) \exp\left(\frac{-E_a}{kT}\right) L^{\frac{1}{\beta}} dt \tag{3.3}$$

It is obvious that by comparing (3.2) and (3.3), the maximum TDDB damage D an interconnect wire accumulates before failure is $D_{max} = D(t = TTF) = 1$ under this definition. The rationale behind (3.3) is that TDDB of the interconnect wire can be presented by a linear accumulation of 'TDDB Damage' along the wire, and that the accumulated

TDDB damage exceeding a certain threshold leads to failure of the interconnect. Note that similar simplification of linear 'damage accumulation' is widely used in reliability analysis works [42, 36].

Note that E is usually non-uniform along the perimeter of interconnect wires due to layout effects and IR drops, and E can even be time-variant. Furthermore, the same applies to T as well. So the layout-dependent TDDB damage for an interconnect wire in such complicated cases should be calculated through an integral on the perimeter (L) of the wire over time:

$$D_{wire} = \frac{1}{K'} L^{\frac{1}{\beta}-1} \int_0^t \oint_L \exp\left(\gamma \sqrt{E(l,t)}\right) \exp\left(\frac{-E_a}{kT(l,t)}\right) dl dt \tag{3.4}$$

In order to locate TDDB hotspots, which are interconnect wires with high failure risk, it is sufficient to assess the TDDB damage accumulated over a set of all interconnect wires. To simplify the problem further, static E is assumed as the more complicated time-variant E-induced failure can be mapped to static E through measurements as done in [3, 33]. Similarly, steady-state or worst-case temperature is considered as TDDB is a long-term failure effect. As a result, the constant terms in (3.4) can be removed for a simplified metric *Damage rate (R)*:

$$R = L^{\frac{1}{\beta}-1} \oint_L \exp\left(\gamma \sqrt{E(l)}\right) \exp\left(\frac{-E_a}{kT(l)}\right) dl \tag{3.5}$$

It is also assumed that temperature is uniform across the layout for simpler calculation. For non-uniform temperature, the later introduced partition based approach can be used to mitigate this assumption, in which constant temperature is assumed in one

partition.

$$R = L^{\frac{1}{\beta}-1} \oint_L \exp\left(\gamma\sqrt{E(l)}\right)dl \tag{3.6}$$

The most vulnerable wires would therefore be identified by calculating and comparing this metric for each wire.

3.2.3 Lifetime of chip

As lifetime of each wire has a Weibull distribution with shape parameter β. The survival rate S (chance of not failing) and lifetime TTF of a wire are [55]:

$$S(t)_{wire} = \exp[-(t/\gamma)^{\beta}]$$

$$TTF_{wire} = \gamma \cdot \Gamma(\frac{1}{\beta} + 1)$$

Given the lifetimes of individual interconnect wires, the lifetime of the full chip can be derived by multiplying the reliability of each wire based on the assumption that failure of any single wire results in a short circuit and thus failure of the entire chip. Therefore, the reliability of the chip is:

$$S(t)_{chip} = \prod_i S(t)_{wire\_i} = \exp[-\sum_i (\frac{t}{\gamma_i})^{\beta}]$$

By comparing the relationship between S and TTF, lifetime of the chip in terms of BEOL TDDB is derived [3]:

$$TTF_{chip} = (\sum_n \frac{1}{TTF_n^{\beta}})^{-\frac{1}{\beta}} \tag{3.7}$$

Note that the assumption that one failed wire fails the chip is made for simplicity. If redundancy is used in design, the calculation of S can be adjusted accordingly to incorporate the redundancy.

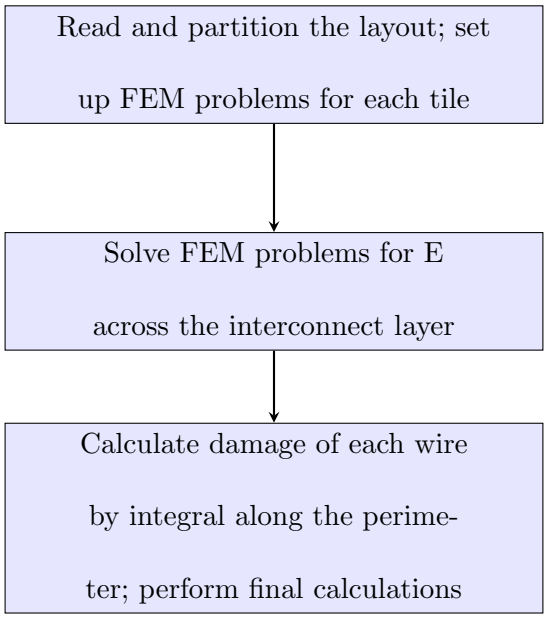


Figure 3.2: The three key steps of the analysis flow

3.3 Layout partition-based TDDB wire damage analysis

This section introduces in detail the proposed approach to solving electric field across the layout, calculating the TDDB damage accumulation rate of interconnect wires, and locating the TDDB hotspots. An overview of the three key steps of the analysis flow is given in the flowchart in Fig. 3.2.

Basically, the proposed full-chip TDDB analysis flow first partitions the layout into many smaller tiles and records which tile each interconnect wire belongs to. After this, FEM problems for each tile are set up with the appropriate boundary conditions and solved for the electrical field distributions. Then, the TDDB damage accumulation rate described by (3.6) can be calculated. Finally the wires with maximum TDDB damage accumulation rate can be found and marked as the *hotspot* wires. Furthermore, lifetime of the chip can be calculated using (3.7).

In the following, additional details of each step are given.

3.3.1 Layout partition

As (3.6) shows, magnitude of electric field is required across the full layout in order to evaluate the damage accumulation rate for each interconnect wire. As discussed previously, 3D geometries are simplified to 2D for this work, and thus different layers are analyzed separately. However, it is still too computationally expensive to conduct FEM analysis of a full interconnect layer. Therefore, it is impractical to solve electric field over an entire interconnect metal layer in one FEM problem. One important point that is helpful for simplifying the problem is that in each metal layer, most interconnect wires are mainly routed in the same direction. For example it is horizontal for M3, and vertical for M4. As a result, the voltage applied on one wire is unlikely to contribute to the electric field several channels away as there will be a high likelihood of other interconnect wires in between, 'blocking' the electric field lines. This makes it reasonable to partition a big layout into smaller tiles, solve FEM problems in these tiles of much smaller size, and combine the results at the end. Fig. 3.3 is an example showing this idea. All the blue lines are interconnect wires of M3 in the layout. Details about this layout will be further discussed in Section 3.4. The dashed yellow lines in the figure show the partition of the layout into much smaller tiles. Electric field in each tile would be solved as if there is no other wire outside the tile.

Through experimentation (explained further in Section 3.4), it is discovered that the electric field solved in the center area of the tile is accurate. This is reasonable as all surrounding wires which influence the electric field at the point of interest are included in the tile and thus the results are expected to be correct. However, in some cases, electric

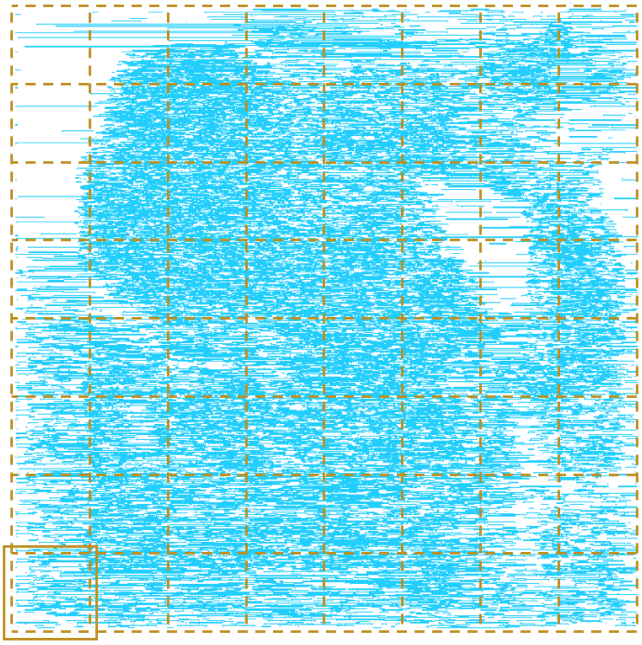


Figure 3.3: Full layout of M3 and the partition scheme

field around the boundaries of the tile is inaccurate as the adjacent wires in neighboring tiles are not included in the tile. This issue can be solved by enlarging the tiles. This idea is also shown in Fig. 3.3, where the yellow solid rectangle at the lower left corner represents the first tile after enlargement. Basically, the area analyzed for each tile is enlarged and the original tile is now the region where results are deemed as valid (the 'effective area'). The tile perimeter expansion ratio can be small since only close-range adjacent wires are required to be included in the extended area. Wires that are farther than, for example, 10 times the feature size or pitch can be safely ignored as they are unlikely to contribute to the electric field inside the tile. In comparison, the size of the effective area would be much larger than that of the feature size. Thus, enlarging the tiles is not expected to significantly affect the analysis runtime.

A library called python-gdsii [45] is used to collect geometry information of all wires on each layer. Once the size of each tile is defined, geometry of wires in each tile is extracted for the next step. At the same time, it is recorded which tile each wire is located in to assist step 3. One challenge is that the wires are broken into segments in the gdsii file, and there is no information indicating which segments each wire should be composed of. In other words, wires in complicated shapes need to be built back by locating and concatenating segments in simpler shapes such as rectangles. To accomplish this, an algorithm determining segments overlaps and union find [23] are used. Note that LVS (Layout versus Schematic) could also be used to analyze the connectivity for all wire segments initially, thus saving the time for running the union find algorithm.

3.3.2 Solving E and TDDB damage in each tile

In this step, both the geometry information for the wires inside each tile and the voltages on them are needed to set up the FEM problem. The geometry information is available after the first step. The voltages are set arbitrarily in order to simulate the worst case: all wires in horizontally-routed layers are sorted by their y-coordinates, while those in vertically routed layers are sorted by their x-coordinates. The wires are then alternately set to VDD and GND. Although the electric field computed in this manner would be higher than would be expected in real applications, this approach is sufficient to identify vulnerable wires. Essentially, a wire is vulnerable to TDDB failure due to sustained high electric fields produced by higher voltage, higher temperature, narrower space, longer length, or over time. Hence, the worst-case electric field can be used to find the wires at risk. Note that if this evaluation flow can be implemented in standard sign-off flow, where signal pattern

for each network is available, more realistic boundary conditions of voltage can be applied, resulting in more trustworthy results and a more accurate determination of time to failure. In [33], similar work has been done and it shows that the worst case time to failure is about half of the realistic one for wire pairs with 50% duty cycle. Thus, if given more voltage information, simply adding a coefficient like 2 to lifetimes of wires of different groups can result in a more accurate estimate of time to failure.

In this work, all wire segments are treated as straight lines, so the line roughness effect (or equivalently, the variation of spacing) has been ignored. Note that this problem can be solved by adding a coefficient to the final result of time to failure as is done in [65].

COMSOL is used to set up and solve the FEM problems. TDDB damage accumulation rate of each wire as described in (3.6) is calculated through an integral in COMSOL. At the same time, the length of each wire in the tiles is also recorded for the next step. Thus, the results obtained in this step for each tile are the damage accumulation rate R and length L of each wire in the tile.

Note that all these calculations are done within the scope of one tile, so calculations in different tiles can be carried out in parallel.

3.3.3 Integral of long wires for final results

As indicated previously, there may exist long wires that span more than one tile. In the previous step, the length and damage accumulation rate for these wires are available separately for each tile that they span. Another step of merging the integral results from different tiles is needed to get the final R values. Then the damage accumulation rates for

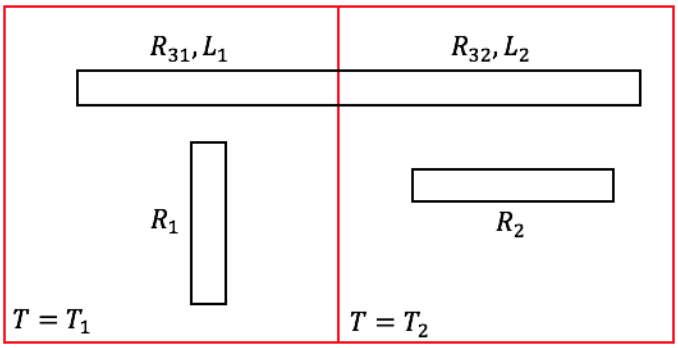


Figure 3.4: An example of a wire crossing a tile boundary, with two tiles and three wires in total

all wires are available and thus, so one can identify the most vulnerable ones. In Fig. 3.4, an example is given to show how long wires are dealt with in the simple case where there are only two tiles and three wires. Wire 1 lies completely within the left tile and wire 2 lies completely within the right tile. Wire 3 lies across both tiles. Note that the temperature is different in the two tiles so (3.5) shall be used to calculate R. Solving electric field distribution and calculating integral of damage accumulation rate R gives results of R_1, R_{31} in the left tile, and R_2, R_{32} in the right tile. R_1, R_2 are final results of the corresponding wire as they lie in only one tile. However, there is an additional step needed to get R_3:

$$R_3 = \left(\frac{R_{31}}{L_1^{\frac{1}{\beta}-1}} + \frac{R_{32}}{L_2^{\frac{1}{\beta}-1}}\right) * (L_1 + L_2)^{\frac{1}{\beta}-1} \tag{3.8}$$

in which L_1, L_2 denotes the length of the part of wire 3 in each tile respectively, with $L_1 + L_2$ being the total length.

Note that if there are errors due to electrical field accuracy, E, as shown later in Table 3.1, this error is not accumulative across partitions. The reason is that in (3.8), the damage rates computed from each partition R_x are added to together to compute the final

damage rate. As a result, if each partition has 2% error (larger or smaller), then the total error in damage rate will be smaller than 2% as errors will not be amplified along different partitions.

To conclude, Algorithm 2 lists the details of the three steps discussed.

Algorithm 2 Full analysis flow

1: Step 1 (Section 3.3.1):
2: **for all** Tiles **do**
3: Find wire segments in tile Merge segments back to wire Record long wires Set voltage on wires
4: **end for**
5: Step 2 (Section 3.3.2):
6: **for all** Tiles **do**
7: Initialize FEM problem Solve FEM problem for E Compute R and length of each wire with (3.6)
8: **end for**
9: Step 3 (Section 3.3.3):
10: **for all** Long wires **do**
11: Merge R from different tiles with (3.8)
12: **end for**
13: Compare R of all wires for hotspot wire
14: Compute chip lifetime with (3.7)

3.4 Numerical results

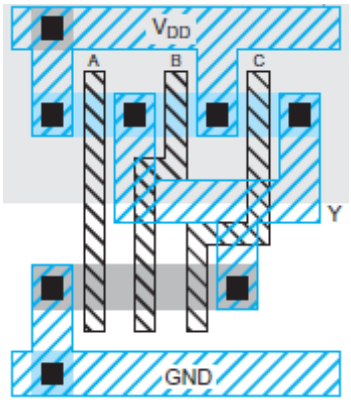
Three sets of simulations are done to validate the proposed analysis flow: the first analyzing a comb structure and a twisted line structure with (3.6) to show the layout effect, the second validating the accuracy of electric field and damage of wires calculated with the layout partition-based approach, and the last demonstrating the analysis flow on an example layout.

A CPU test chip is synthesized for the latter two sets of simulations using a 32nm educational technology. The size of the layout is 193μm by 193μm. Further information about the layout is provided in the second part of the simulation in Table 3.3.

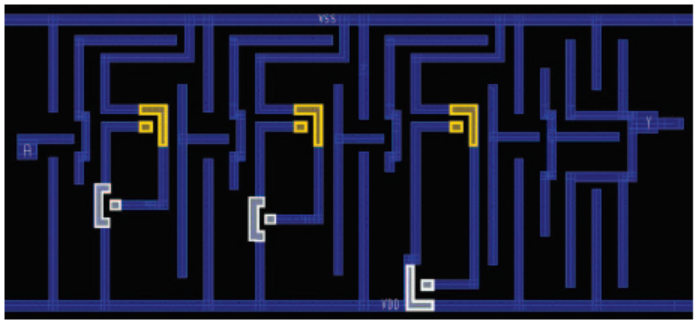
For numerical analysis, the electric field acceleration factor γ used in the equation is 20 if the unit of E is MV/cm, which is fitted based on published experimental data [18]. The Weibull shape factor β is 0.6, which is obtained from 30nm half-pitch test vehicles [18]. Temperature is assumed to be constant. The additional constant $K' \exp\left(\frac{E_a}{kT}\right)$ in (3.4) is required when evaluating lifetime. The value $1s/cm^{\frac{1}{0.6}}$ is used, which is an assumed number for the purpose of demonstrating the lifetime calculation.

3.4.1 Comparison on three structures

In this subsection, the proposed method is compared against other full-chip TDDB methods on two wire structures and show that the electric field calculation is necessary for lifetime prediction. The simulated target is three different layout structures: a comb structure, a parallel line structure, and a twisted line structure. Comb structures usually appear in lower levels of interconnects for the purpose of saving area of logic gates. The two



(a) A NAND gate, courtesy of [66]



(b) A delay cell, courtesy of [33]

Figure 3.5: Comb structures in designs

examples are shown in Fig. 3.5. Multiple comb structures exist between the VDD power rail and signal nets.

The parallel line structure and twisted line structure usually appear at higher metal levels, as the router typically routes interconnect wires in one primary direction for each layer. Fig. 3.8, from the synthesized layout, contains many examples of parallel lines and a few twisted line structures. As shown in Fig. 3.6, the line widths and space between wires are both 60nm, and the total dielectric channel length is 1000nm for both (excluding the corner part in (a) and (c), consistent with the method in [3]). Specifically, the first two structures are simplified from the two commonly-used structures in studies of TDDB physics, namely serpentine-comb and comb-comb structures [7].

In [3], the time to failure (TTF) of a vulnerable wire with length L_v is scaled from the measured test wire structure under same stressing conditions with wire length L_t as

follows:

$$TTF_v = TTF_t(\frac{L_t}{L_v})^{1/\beta} \tag{3.9}$$

where L_v is the only input of this method, which is 1000nm for all three examples (shaded area), so the three structure will have the same lifetime as the wire length are same.

The method introduced by [33] can identify vulnerable spots, which does lead to the correct result that comb structure and twisted line structure fail faster than the parallel line structure. However, length effect is not considered in this method. If the length of stressed dielectric doubles, this method will still give the same result for TTF.

The proposed method takes both layout effect and length effect into consideration. In the proposed method, the electric field in the three structures (shown in Fig. 3.7), R of three wires using (3.6), and obtained the lifetime, are computed, with the help of COMSOL. It turns out that the lifetimes of the comb structure and twisted line structure are only $0.89\times$ and $0.7\times$ that of the parallel line structure respectively. Furthermore, studies [32, 7] show that comb structures are indeed more vulnerable to TDDB failures due to the increased electric field at the tips. Therefore, the proposed method is more accurate than the full chip TDDB method proposed in [3] and it is indeed necessary to base the analysis on electric field distributions for improved accuracy.

3.4.2 Validation of the partition-based method

As discussed in Section 3.3.1, partitioning the layout might lead to some inaccuracy around boundaries as information in neighboring tiles is missing. To reduce loss of accuracy

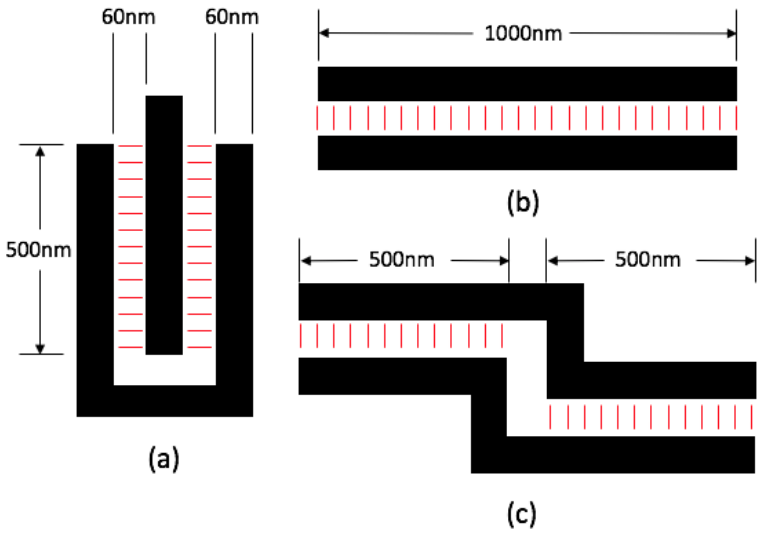
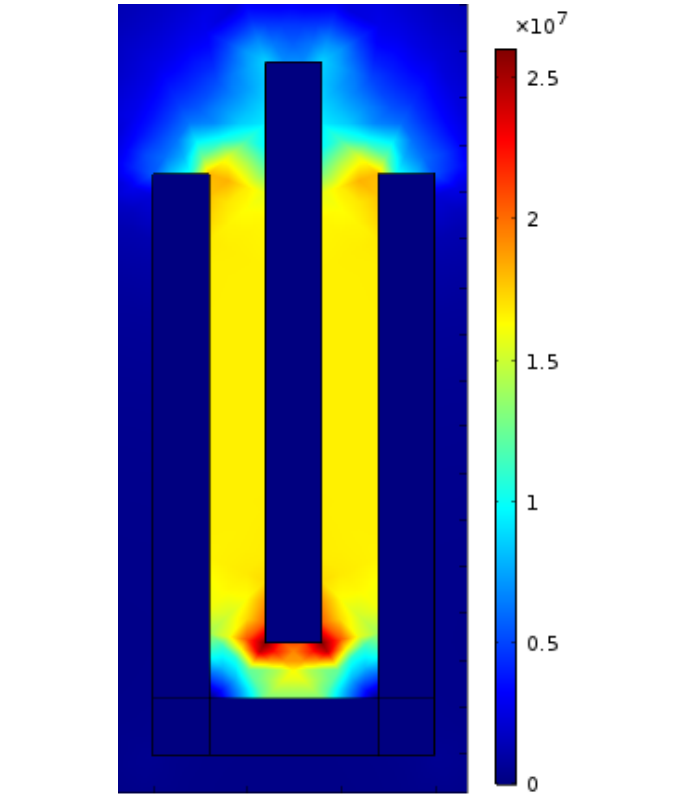


Figure 3.6: Three structures analyzed: (a) comb (b) parallel line (c)twisted line

in the electric field analysis, it is proposed to enlarge the tile. The second set of simulations is conducted to verify the effectiveness of this approach.

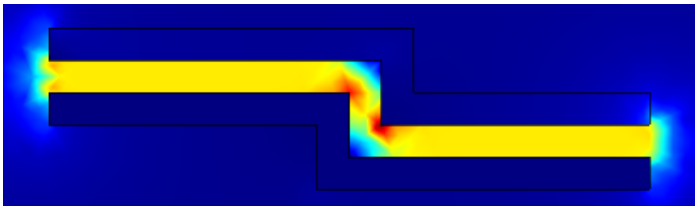
The simulations are conducted on an extracted small tile from the synthesized layout, as shown in Fig. 3.8. Wires in M3 layer of this region are represented by lines in the figure. The size of the entire tile is 1.6μm by 1.6μm. The green square is the enlarged tile to be analyzed, with an edge length of 1.2μm, and the red square is the effective area of the tile, with an edge length of 1μm.

Three simulations with different setups are done. All results are collected within the red square in the figure, which is the effective tile. At first, electric field is solved across the entire extracted tile (the complete Fig. 3.8) and damage accumulation rate of wires is calculated within the scope of the red square. The electric field and damage accumulation rate results obtained in this phase are denoted by $E0$ and $R0$ respectively. These are the



(a)

(b)



(c)

Figure 3.7: Electric field distribution of the three structures analyzed

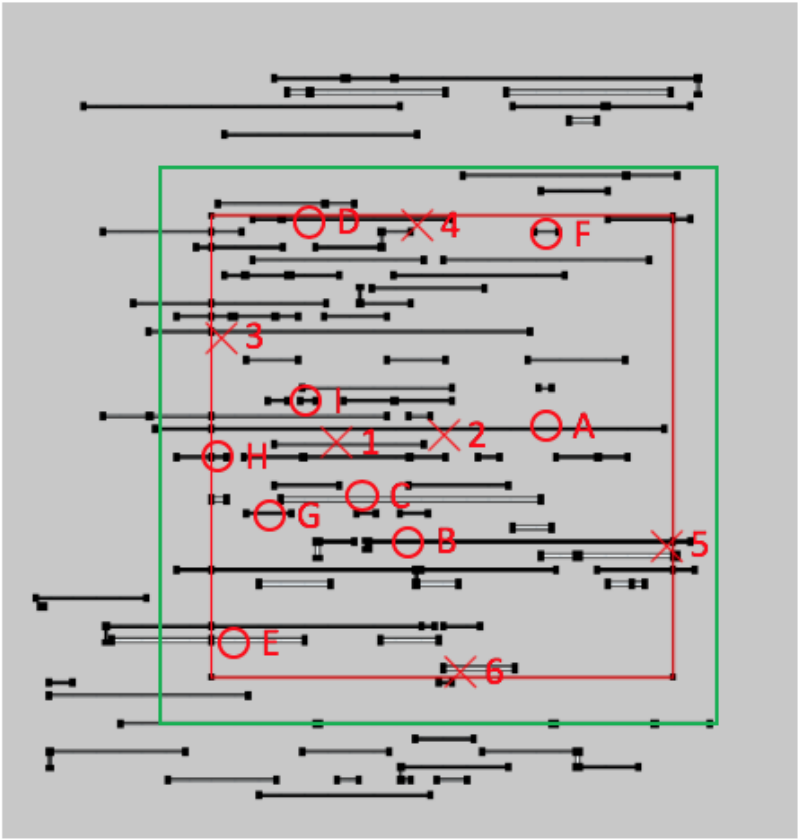


Figure 3.8: The layout tile used to validate the partition-based approach

reference results used to compare with in the other two simulations. Secondly, with the same voltage settings on the wires, the scope of the FEM problem is reduced to the red square. Results from this step are compared with $E0$ or $R0$ and errors are calculated and denoted by $Error1$. As there is no enlargement of the tile being analyzed, the results in this step are expected to show significant discrepancies. Finally, a third analysis is done with FEM problem set to the green square, enlarging the red tile. Similarly, errors of results in comparison with $E0$ or $R0$ are calculated and denoted by $Error2$.

To compare the results precisely, six locations are chosen to evaluate the electric field. They have been marked by red "$\times$" symbols with indices in Fig. 3.8 along the wires of interest. Points 1 and 2 are in the center of the tile, while points 3 through 6 are near the boundary of the effective tile area. They are chosen arbitrarily in order to cover as many variations as possible. Table 3.1 lists E, $Error1$ and $Error2$ at these points. As can be seen from the table, only points 3 and 6 show significant errors. However, the electric fields at these two points are much lower than the other four points. According to (3.6), the damage accumulation rate of a wire is mainly determined by high E locations along its perimeter. Therefore, the errors at low E points are not expected to influence the final results of the damage accumulation rate. The reason for the large $Error1$ values of points 3 and 6 is that the default boundary condition in FEM makes the results around the boundary less accurate. Fortunately, enlarging the tile leads to a significant improvement in the accuracy for these cases, as $Error2$ turns out to be much smaller than $Error1$.

Table 3.1: Results of E at points marked "×" under different setups. Unit is V/m. Error is defined as $(E - E0)/E0$.

Point	$E0$	Error1(%)	Error2(%)
1	4.032E6	0.00%	0.00%
2	1.671E6	-0.24%	-0.33%
3	4.006E4	47.62%	-6.11%
4	3.261E6	0.30%	0.40%
5	3.815E6	5.70%	-0.03%
6	6.471E3	278.37%	-2.51%

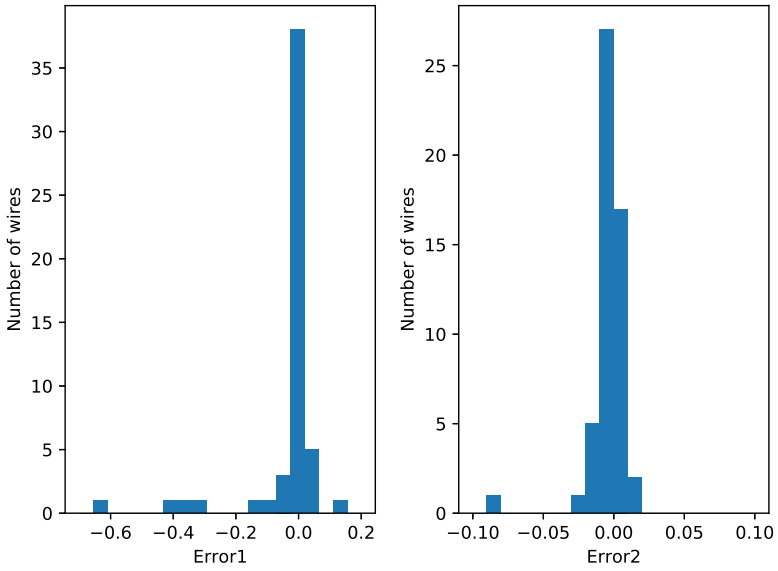


Figure 3.9: Distribution of $Error1$ and $Error2$ of R among all wires

Table 3.2: Results of R of marked wires marked "◯" under different problem setups. Error is defined as $(R - R0)/R0$.

Wire	$R0$	Error1(%)	Error2(%)
A	1.552E-10	-0.24%	-0.11%
B	7.574E-11	0.33%	-0.05%
C	5.599E-11	-0.06%	-0.14%
D	4.538E-11	-63.06%	0.88%
E	1.038E-12	-39.60%	0.00%
F	1.973E-13	-37.72%	0.80%
G	1.198E-13	0.09%	-8.43%
H	2.679E-13	6.50%	-2.87%
I	3.931E-12	1.26%	1.51%

Comparisons on E on several points are not enough, as there are a very large number of points. The accuracy of R of wires is more important for getting the right evaluation of lifetime. As a result, another comparison is done for the R computed in the two cases. The nine evaluated wires are marked by red "◯" along with indices in Fig. 3.8. First, histograms of errors in R of all wires are plotted in Fig. 3.9. The definition of $Error1$ and $Error2$ is same as that in the previous comparison of E. It can be seen that enlarging the tile has greatly decreased the overall errors. Second, several extreme examples in Table 3.2 are listed as done with the comparisons of E. Wires A to C are wires with the highest damage accumulation rate. These wires are the ones of most interest and the table shows that the results for them are well-matched between all approaches. Wires D through F are wires with largest errors for setup 2 (FEM done in the red square). These significant errors are due to the fact that the nearby wires that are in the green square but not in the red square are missing in the FEM analysis. More importantly, R of wire D is not small, which means it can be a potential hotspot wire and thus the discrepancy on this wire is unacceptable. This further proves the necessity of enlarging the tile. Finally, wires G to I are wires with largest errors for setup 3 (FEM done in the green square). Note that the damage accumulation rates for all these wires are comparatively small, indicating that they are not the vulnerable wires that one is most interested in.

In conclusion, the proposed partition-based method, which is the last setup in the simulation, has achieved high accuracy on high-E spots, and high-R interconnect wires, which are the objects of interest in this work.

Table 3.3: Analysis results of all layers

Layer	Number of wires	Max R	Lifetime (yr)	Runtime (h)
M1	53756	3.73E-7	118	11.8
M2	26576	3.47E-8	376	7.17
M3	15977	1.64E-8	1570	4.75
M4	5093	2.65E-8	4110	1.32
M5	2225	1.07E-8	1.49E4	0.87
M6	929	8.73E-9	1.08E5	0.38
M7	480	6.96E-9	1.32E5	0.30
M8	173	2.37E-9	3.42E6	0.12
M9	77	2.28E-9	1.85E6	0.10

3.4.3 Analysis of an example layout

Hotspot detection and full-chip lifetime analysis were carried out with the partition-based scheme on all interconnect metal layers. The number of wires and the damage accumulation rate of the most vulnerable wire in each layer is given in Table 3.3. Computation time for each layer is also listed in the table. FEM analysis takes most of the time for each layer. Lifetimes of each layer calculated with (3.7) are also listed. The lifetime of this synthesized chip calculated with (3.7) is 39.76 years.

M1 turns out to have the shortest lifetime, as opposed to upper metal layers in [3]. This is reasonable as there are many power delivery rails which are long and DC stressed. They are stressed with relatively higher electric fields because of narrower spaces in M1 and higher wire density. Additionally, M1 also has dielectric materials with the lowest dielectric constants, while upper metal layers with large space use more robust materials. Furthermore, the more complicated geometries compared with other layers make the lifetime

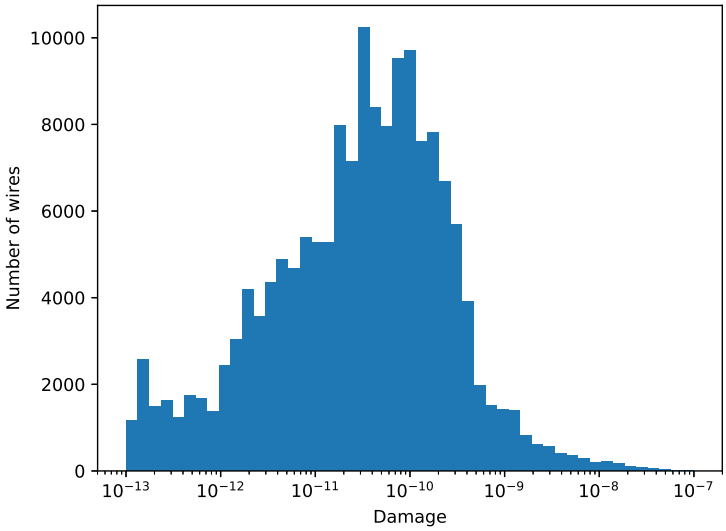


Figure 3.10: Distribution of R among all wires

even shorter because of the aforementioned layout effect. Hence, M1 is expected to have the shortest lifetime in all layers.

As to hotspot detection, Fig. 3.10 shows the distribution of R values of all wires. There are 733 wires with R ranging from 10^{-8} to 10^{-7}, and only 74 with R ranging from 5×10^{-8} to 10^{-7}, which means the analysis flow has identified a small number of hotspots wires from all 105286 wires. These wires would be the first ones that a designer would address in order to improve the BEOL TDDB reliability of the chip. Furthermore, it is possible that these 74 hotspot wires are composed of not only power delivery network wires, but also clock tree wires and signal wires. The latter two categories may be treated differently due to the lower duty factor, which would further reduce the number of serious hotspots.

Figure 3.11: Full M1 layer view with most vulnerable wire highlighted

As an example, a wire in M1 is identified as the most vulnerable in all wires across the chip. The full layer of M1 with this wire highlighted is shown in Fig. 3.11. It can be seen that the wire is actually a long power rail and spans a significant extent of the layout. Results of other interconnect layers also show that the most vulnerable wires are long wires. This is logical as (3.2) has introduced length scaling so that long wires are likely to be more vulnerable under similar stressing conditions. As the wire density in neighboring area along this wire is comparatively high, the integral in (3.6) is comparable to that of other wires, so its longer length contributes to its greater vulnerability.

3.5 Summary

A layout partition-based interconnect TDDB analysis flow has been proposed in this paper. Time to failure is evaluated by a newly-defined metric called TDDB Damage defined on each interconnect wire, and by comparing damage accumulation rates, hotspots (vulnerable) wires at high risk of TDDB failure) are identified. Based on electric fields analyzed from FEM, calculation of damage accumulation rate is conducted in each partitioned tile, and final results are obtained by merging and comparing intermediate results from small tiles. Since the approach is based on solving for electric fields, it can cover various electric field acceleration models commonly used and can also account for the non-uniformity of electric field in all layout patterns. The new method compares favorably in terms of accuracy against a recently proposed full chip TDDB method. This allows a better determination of risk and helps avoid overly pessimistic designs due to larger-than-required interconnect spacing.

Chapter 4

Data-Driven Fast Electrostatics and TDDB Aging Analysis

A significant shortcoming of the method introduced in chapter 3 is the long computation time required to solve the FEM problems. They are set up from partitioned layout tiles with identical boundary conditions; they conform to the same physics law, and go through the same solving process. As data-driven deep learning approaches provide new perspectives for learning the physics-law and representations of the physics dynamics from the data, this inspires the use of machine learning to speed up the solving process.

The work presented in this chapter aims to accomplish this. A data-driven learning based approach for fast 2D analysis of electric potential and electric fields based on DNNs (deep neural networks) is introduced. Originally aimed to speed up the FEM solving process in TDDB analysis, the proposed method is essentially solving electrostatics problems under certain fixed boundary conditions. It is applicable to problems involving electrostatics other

than TDDB. The proposed method is based on the observation that the synthesized VLSI layout with multi interconnect layers can be viewed as layered images. Image transformation techniques via CNN (convolutional neural network) are adopted for the analysis. Once trained, the model is applicable to any synthesized layout of the same technology. Training and testing are done on a dataset built from a synthesized CPU chip. Results show that the proposed method is around 138x faster than the conventional numerical methods based software COMSOL, while keeping 99% of the accuracy on potential analysis, and 97% for TDDB aging analysis. This work is published in [51].

4.1 Overview

Electrostatics is an important subject of study as it is pivotal in many VLSI modeling applications. The goal is to compute electric potentials and electric fields with some given voltage and current sources (boundary conditions). In the back-end of VLSI chips, strong electric field can induce failure of the dielectrics, which is known as TDDB [44]. Simulation of this aging effect requires electrostatics. Also, several methods of parasitic extraction involve electrostatics simulations in the chip layouts [15, 72]. Furthermore, global placement is also proposed to be modeled as electrostatic problem recently [41].

Traditionally, electrostatics is primarily solved by numerical methods, with spatial discretization of the governing equation. Such numerical methods typically require a mesh of the complicated layout or geometry, which can be computationally prohibitive for large designs.

Recently, machine learning, especially deep learning has revolutionized fields such as image, text, and speech recognition [38]. These fields require statistical approaches which can model nonlinear dynamic functions very well. Deep learning has shown potential and promise in practice for such tasks. But using DNN to learn the spatial and temporal dynamics in physics laws is a less explored field. Some early works have been proposed recently [62, 63, 73] to solve the partial differential equation based on supervised learning. However, the problems those methods solve are too small to be practically significant.

Inspired by these observations, this work proposes a new data-driven learning based approach for fast analysis of electric potential and electric fields, based on which TDDB aging analysis can also be done efficiently.

4.2 Related works

In convention, electrostatics problems are solved using discretization methods such as FEM or finite difference method (FDM) [72]. Results from commercial tools based on FEM such as COMSOL are usually deemed golden. However, the speed of these methods is limited. In conclusive words, FEM first discretizes the domain to be solved by a mesh. The mesh and the shape function chosen together define a function space. Any function in the mesh can be approximated by linear combinations of the shape functions with appropriate expansion coefficients. A numeric solution to the original PDEs can be then found by searching in this function space for the one that best fits the original equations. This is done by setting up and solving a linear system from the original PDEs, with the expansion coefficients to be solved. Typically, the final linear system is composed of tens of thousands

of unknowns, known as DOF (degree of freedom). Solving a problem of this scale is not very expensive, yet is still significant in a longer routine.

Recently, deep learning is gaining a lot of attention and has been explored in some practical simulation problems, mainly because of its speed once the model is well trained. Tang et al. [62] proposed to solve Poisson's equation, specifically in the case of electrostatics, using modern CNN-based model. The setup of the problem is similar to this work, where the PDEs (partial differential equations) remain unchanged (Poisson's equation), and the input variables are locations of excitation sources and distribution of permittivity (a coefficient in the PDE). Error was reported as low as 3%. However, the problem size is relatively small (the grid size is 64×64 with no extension). The similar method was also used by Zhang et al. [73], where charge distribution and boundary condition (voltage) are used as the inputs of the neural network. A regular grid is used in all these works. Tompson et al. [63] try to deal with a more complicated time-dependant fluid flow problem. Yet convolutional network is used to speed up some steps in the overall simulation instead of solving the whole problem on its own.

This work also aims to use CNN-based network to solve electrostatics. The problem size is comparatively larger, and it can be further expanded through layout partition.

4.3 Preliminaries of Electrostatics and TDDB

Electrostatics studies the distribution of electric potential and field in cases where the charges are static, i.e., there is no electric current. Electric potential is governed by the

first equation of the Maxwell's equations, also known as Gauss's law:

$$\nabla^2 u = \frac{-\rho}{\epsilon} \tag{4.1}$$

where u is electric potential, ρ is static charge density, and ϵ is the permittivity.

The equation usually comes with the following Dirichlet and Neumann boundary conditions:

$$\begin{aligned} u &= f(x),\ x \in \Gamma_D, \\ \nabla u \cdot \vec{n} &= g(x),\ x \in \Gamma_N, \end{aligned} \tag{4.2}$$

where Γ_D is the part of the boundary where Dirichlet (voltage) boundary conditions are given, Γ_N is the part of the boundary where Neumann (electric field or current) boundary conditions are given, u is the unknown potential to be solved, $f(x)$, and $g(x)$ are given voltage sources and electric field (or current sources) at the boundaries.

In cases where static charge is absent, which this work focuses on, (4.1) becomes Laplace equation:

$$\nabla^2 u = 0 \tag{4.3}$$

With the solution of u, distribution of electric field is usually obtained by calculating the gradient as per its definition:

$$\vec{E} = -\nabla u \tag{4.4}$$

With the solution of $\vec{E}$ available, one can calculate the TTF of wires by taking the reciprocal of the equation of reliability (3.6):

$$TTF \propto \oint_L \exp\left(-\gamma\sqrt{E(l)}\right) dl \tag{4.5}$$

where γ is a coefficient fitted from experiment data, and L is the perimeter of the wire. By finding the wire with the minimum TTF, the hotspot of the designed layout can be picked out for further optimization.

Note that in this application concerning electric fields, it is sufficient that only two voltages are involved: VDD and GND. VDD is set to 1V for the ease of computation.

4.4 The proposed data-driven electrostatic analysis

This section introduces the encoding scheme of the electrostatics problem, along with the training and test data generation process, and details of the neural network used.

4.4.1 Problem formulation

As introduced above, this work aims to solve the distribution of electric potential and electric field in dielectrics in the VLSI back-end-of-line. Typical VLSI designs have dimensions in the order of hundreds of micrometers to millimeters. Interconnect wires that are as wide as only nanometers form complex geometries in such large area. This problem size poses the first challenge to feed this problem into a neural network. A typical input image in modern machine learning research [59] has a size of 224 pixels by 224 pixels, which is far coarser than the required complexity of VLSI back-end-of-line.

Layout partition, introduced in section 3.3.1, helps reduce the complexity of the problem. The layout is partitioned into smaller tiles and each tile is analyzed separately, also as shown in Fig. 3.3.

Another challenge is to encode the tiles so that it fits machine learning. The chip layout is originally stored in gdsii format, in which the shapes and positions of elements are stored in a binary format. An easy approach is to convert it into images. In our problem, there are two sets of inputs, namely geometry information of the interconnect wires and the voltage boundary conditions. Note that in the two targeted electrostatics applications introduced previously in Section 4.3, two voltages, VDD and GND, are sufficient for the analysis. Also, the routine used in reliability analysis introduced in chapter 3 that VDD and GND are set alternately on wires is followed here. An important result is that every wire is set a voltage. Consequently, the locations of VDD wires and GND wires are sufficient to describe the problem, as the locations where voltages are set indicate the existence of wires. Also, the lifetime analysis is done on a worst case scenario. A simple value of $1V$ can be assumed for VDD as more complicated cases involving duty cycles and other voltage values can be simply scaled by adding coefficients [33]. The two sets of boundary conditions (VDD and GND) are encoded in different channels in the encoded image. For each pixel, the value of a channel is set to 1 if the corresponding voltage boundary condition is set at this pixel, and 0 if not. The corresponding size of a pixel can be set to the pitch size of the technology (60nm in this work), as it is basically the unit length in the geometry. The results of electric potential distribution can also be displayed as images, using the same method. This encoding is demonstrated with an example shown in Fig. 4.1 and Fig. 4.2.

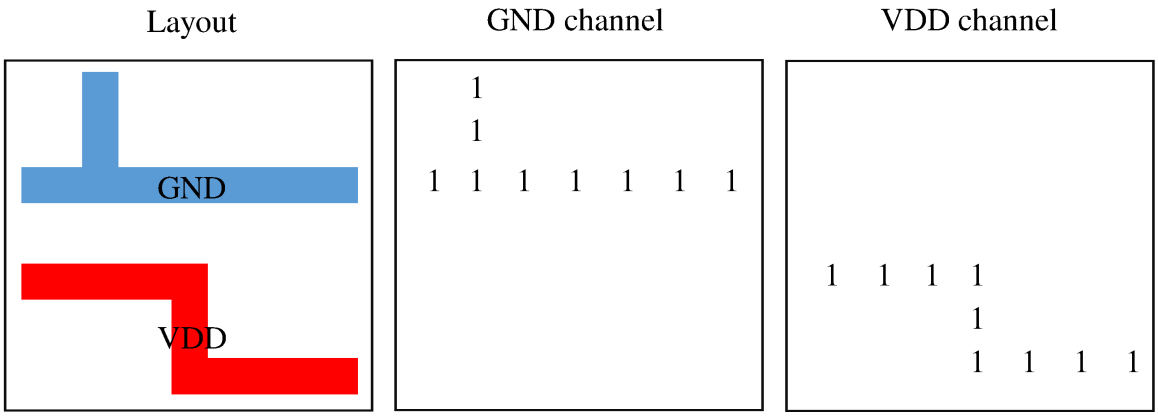
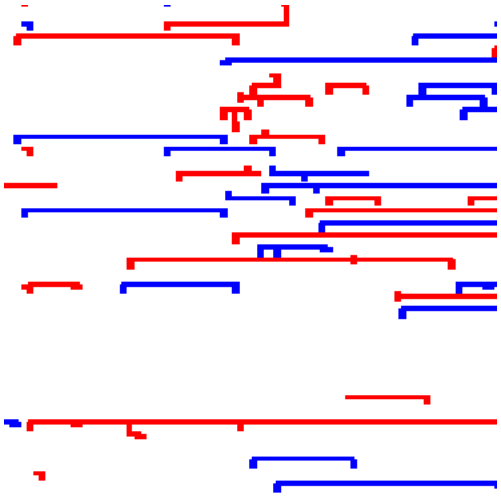
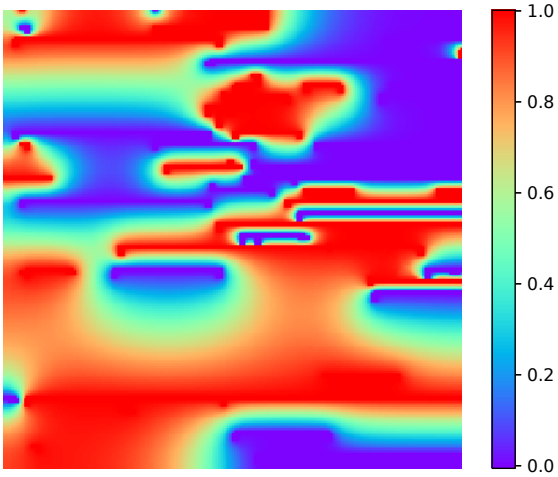


Figure 4.1: Encoding a layout tile into image



(a) VDD wires: red; GND wires: blue



(b) Solution of electric potential (unit: V)

Figure 4.2: An example of encoded tile image of and its corresponding solution of potential

4.4.2 Structure of the neural network

The overall architecture of the neural network is shown in Fig. 4.3. The hyper parameters are listed in the figure as well. Also, the output tensor sizes of each layer are listed in square brackets. The number of channels are listed last in the tensor sizes. The structure is very similar to autoencoders [28] as well as U-Net [56]. It is composed of a contracting network, called encoder, and an expansive network, called decoder. Skip connections [31] are added between the encoder and decoder layers that have same output size, shown as dashed lines.

First, the encoded layout tiles are expanded to 256×256 pixels sized square images by padding zeros on four sides. Then they are sent into the first convolutional layer, composed of four convolution kernels with sizes of 3×3, 5×5, 7×7, 9×9 respectively. They are used to capture features nearby, however in different distances. The four feature maps are concatenated and sent to the next layer. The following convolutional layers use single 5×5 convolutional kernels. All convolutional layers are followed by max pooling layer to down-sample the tensor. Through the encoder network, the image is down-sampled to 1×1×512 vector, which can be seen as latent features of the layout tile extracted by the encoder. Then, the remaining part of the network, namely the decoder, reversely up-samples the latent feature vector to the output image, which is the solution of electric potential, or u.

Instead of traditional transposed convolutional layers, the decoder uses resize-convolution blocks [49] to up-sample the latent vector, which helps avoid checkerboard artifacts in the final image and thus increases the accuracy.

Another technique used in the network is adding six skip connections [31] between the encoder and decoder layers that have same output size, shown as dashed lines in Fig. 4.3. The intuition of adding such connections is that u always decrease gradually from VDD wires to GND wires in studied problems. In other words, the final voltage map is highly related the known voltages that are set on wires. So the information extracted by the first convolutional layer, which should include whether a VDD or a GND wire is nearby, is a strong deciding factor to the final output. The outputs from the first convolutional layer are used as a founding "big picture" of the final voltage map. The remaining layers of the autoencoder are then expected to learn to calculate more intricate details of the result. They help increase the model accuracy and accelerate convergence in training. Performance of the skip connections will be further discussed in detail in Section 4.5.

Overall, once trained, the network is expected to work like a finite difference method based solver, which is able to output the electric potential distribution from the given interconnect geometry and voltage. Since all layouts synthesized with the same technology have the same pitch size, they can all be analyzed efficiently with the trained model. In other words, the model is universal as it applies to any layout of this technology.

4.5 Numerical results and discussions

The aforementioned neural network has been implemented with TensorFlow 2.0 and Python 3.7. All the following experiments are run on a Linux server equipped with 2 Intel Xeon E5-2698v3 2.3GHz processors and a Nvidia Titan X GPU.

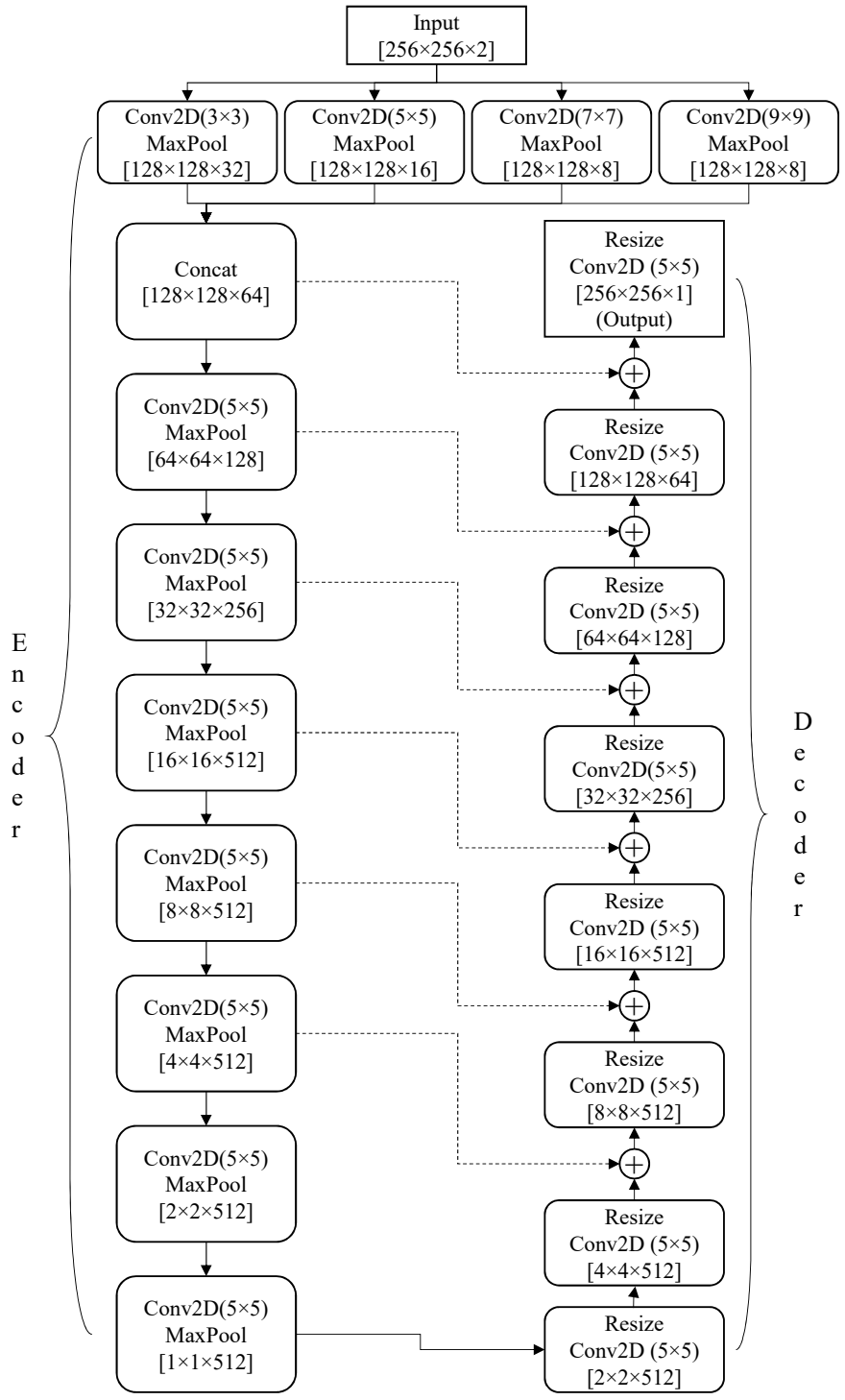


Figure 4.3: Structure of the used neural network

4.5.1 Data preparation and training

This work is done with an example layout synthesized with a 32nm educational technology. The size of the layout is 200μm×200μm, and the interconnect layers are partitioned into tiles of 12μm×12μm. As a result, the grid size is 60nm ($12000nm \div 200$). This value is chosen as the pitch length (minimum distance between interconnect wires) is 60nm.

Example tiles are extracted from two interconnect layers, specifically M3 (Metal 3) and M4, as they have horizontal and vertical routing directions respectively. In total about 12000 tiles are collected as data samples for training and testing the model. COMSOL is used to simulate electrostatics on them for distributions of electric potential and electric field.

As tiles are highly overlapped with each other, it is not reasonable to randomly divide the dataset into training, validation and test set. Otherwise, for any tile from the test set, it is highly possible that there is another tile that is highly overlapped with it in another, say training set, thus defying the test. To overcome this problem, the dataset is divided by region. As shown in Fig. 4.4, the test set is composed of the layout tiles that come from 0≤y≤36μm, the validation set is composed of those from 36μm≤y≤72μm, and the training set includes all the remaining tiles. Furthermore, tiles that cross the dividing horizontal lines y=36μm and y=72μm are dropped from the dataset to avoid the overlap problem between data sets. Overall, 18% of the tiles are used as validation set, another 18% are used as test set, and the rest are training set.

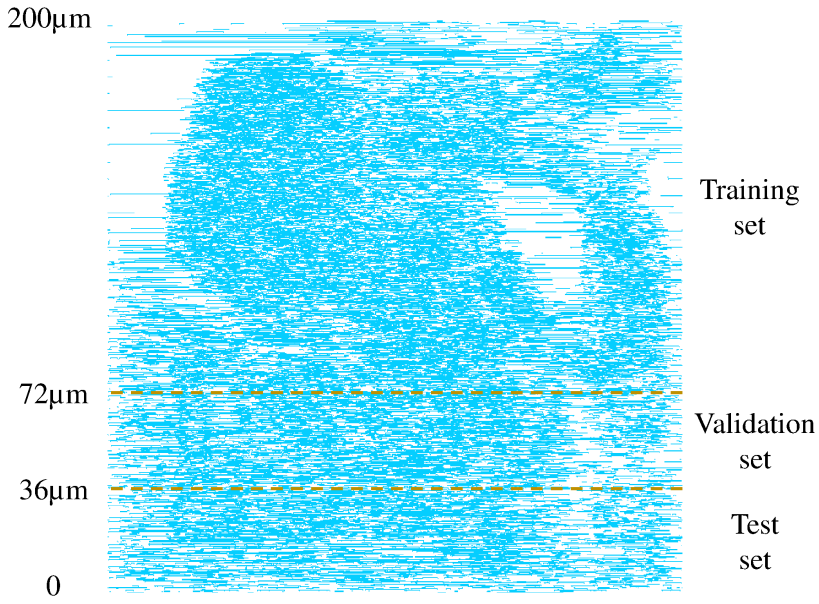


Figure 4.4: The split of the dataset from layout (M3 shown here)

In training, the root-mean-square error (RMSE) is used as the loss function:

$$RMSE = \sqrt{\frac{\sum_{x=1}^{W}\sum_{y=1}^{H}[u(x,y) - u_0(x,y)]^2}{W \times H}} \tag{4.6}$$

where u and u_0 are output electric potential and golden result simulated with FEM respectively, W and H dimension are sizes of the tile. Adam optimizer with the learning rate of 0.00001 is used. The training runs for 350 epochs. Progress of training and validation losses are shown in Fig. 4.5. There is no sign of over-fitting till the end of training. This training process takes in total 12 hours.

The final model after 350 epochs of training are tested with the test set. The RMSE of all test samples is illustrated as histogram in Fig. 4.6. The average RMSE across the whole test set is 0.01, As the range of potential is 0V to 1V in this problem, so the RMSE value is equivalent to error in percentage, which means the average error of the model is

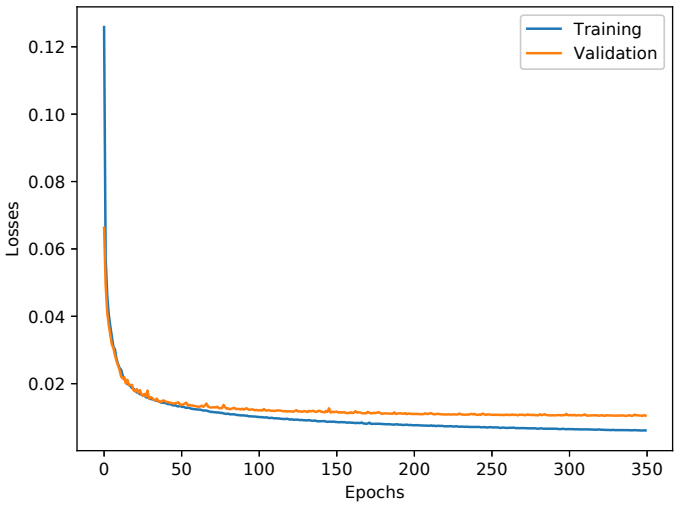


Figure 4.5: Training curves

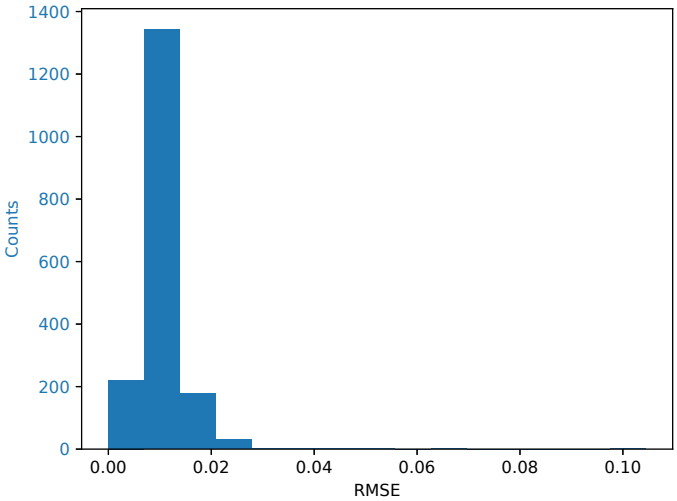


Figure 4.6: Distribution of RMSE over the test set

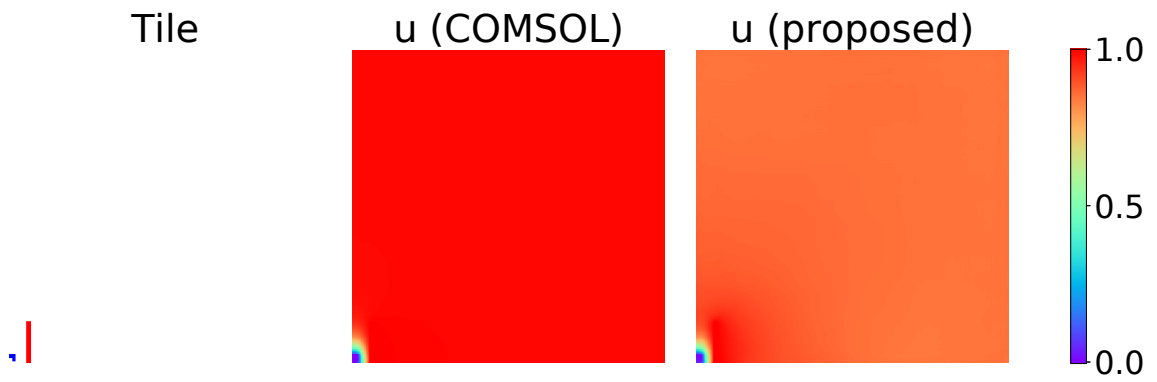


Figure 4.7: The test sample with the largest RMSE. The tile is shown using the same color convention as in Fig. 4.2a

1%. This is very close to the accuracy achieved in training and validation. The maximum RMSE is 10.4%. Only 5 out of all 1760 test samples have errors larger than 5%. It is worth noting that these 5 samples turn out all to be extreme cases. The sample of largest RMSE is shown in Fig. 4.7. It can be seen that this is an extreme case where wires only exist in corners, and it is the other empty areas that contribute to the error. However, these empty areas are actually not studied, as all electrostatic problems focus on wires and their nearby dielectrics. Furthermore, these test samples of extreme cases would not appear in practical applications if layout partition is used, because the related wires would already have been solved in neighboring tiles that have these wires located away from the tile boundaries and thus can achieve more accurate results.

4.5.2 Results of electric potential analysis

To further demonstrate the potentials calculated using the trained model, 4 samples, 2 from M3 and 2 from M4, have been randomly picked from the test set. The calculated distributions of potential are plotted and compared against COMSOL simulation results,

in Fig. 4.8. The tiles are also shown in the figure. The RMSE of these samples are 0.011, 0.013, 0.008, and 0.007 respectively. It can be seen that the electric potential calculated by the proposed method matches the golden results very well for the given four example samples.

4.5.3 Results of electric field analysis

With the solution of potential, electric field can be easily derived by numerically calculating gradients depicted by equation (4.4). In practice, it is calculated through the gradient function from the numpy library. The calculated results are plotted and compared with COMSOL results, shown in Fig. 4.9. The same 4 samples from test set are used for comparison. The largest RMSE of the eight plots are 0.01276MV/cm, with the range being -0.0818 to 0.0825MV/cm, so the error is equivalent to 7.8%. Although the overall distribution is matched well as shown in the figure, it is obvious the results derived using the proposed method appear noisier. Furthermore, it can be seen that the values at corners of wires show some discrepancies. The reason of this is mainly that the gradient at such positions is in fact singular in problems of Poisson's equation. Thus, its value is highly related to the mesh used. In spite of the mentioned problems, the overall error, as mentioned, is within 8%.

4.5.4 Results of TDDB aging analysis

The derived solutions of electric field are used to analyze TDDB aging using equation (4.5).

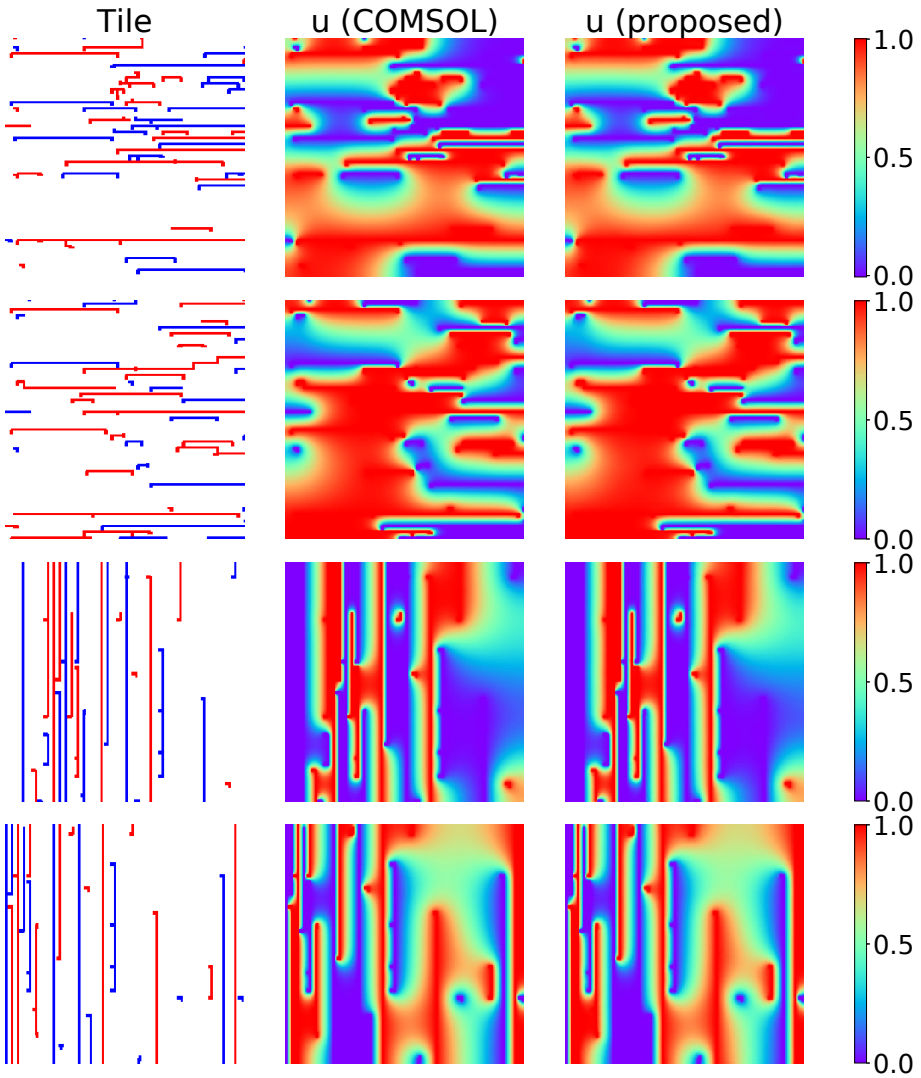


Figure 4.8: Results of electric potential of sample tiles (left) from COMSOL (middle) and the proposed method (right)

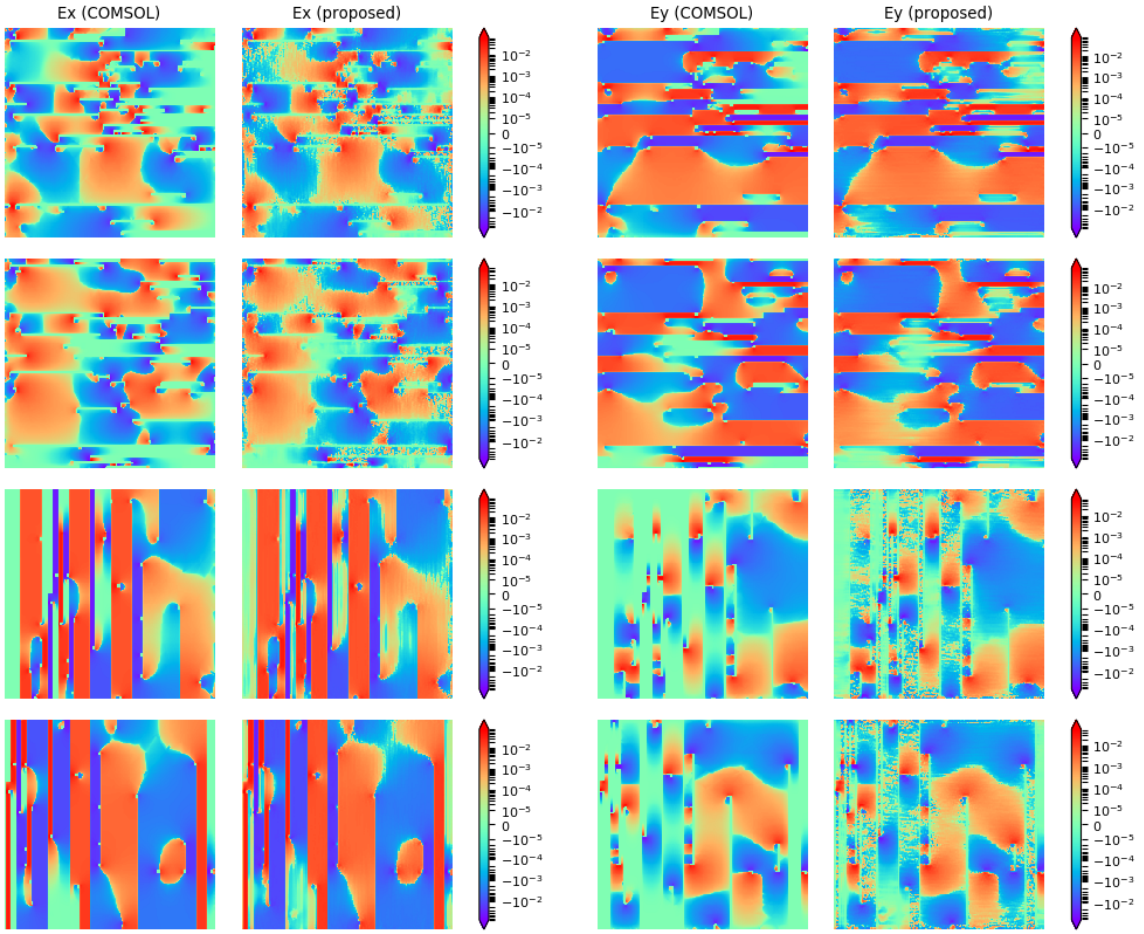


Figure 4.9: Results of electric field of sample tiles from COMSOL and the proposed method. The unit of electric field is MV/cm.

Two sample wires are chosen from each sample tile used in the previous comparison, one set to VDD and one set to GND, and the integral results on them are again compared to those from COMSOL. Table 4.1 lists these results. Note that they are direct integral results so they are expected only to be proportional to the final capacitance values or lifetimes. The differences are calculated with COMSOL results treated as golden reference. The differences between results from the proposed method and from COMSOL are are all within 6.71%, and the average of their absolute values are 2.57%, which is equivalent to 97.43% accuracy.

It is worth noting that COMSOL uses a finer mesh in the simulation, which can help improve the accuracy of numerical integrals, which is similar to the cause of discrepancy in electric fields themselves. Furthermore, the proposed method approximates the geometry of wires with coarse grid meshes. Compared with the original layout that has a few elements that have lengths that are not multiples of the technology pitch size (60nm in this work), mostly because of synthesized vias, the proposed method approximates them onto the closest multiple of the technology pitch size. This is another significant factor leading to errors in the final results of integrals.

Despite all the simplifications and comparatively larger errors in electric field analysis results, the proposed method is still able to achieve around 97% accuracy for the application of TDDB aging analysis.

Table 4.1: Integral results in capacitance and aging analysis of sample wires, calculated using the proposed method and COMSOL

	COMSOL	Proposed	Diff
Wire1	3.486e-5	3.492e-5	0.19%
Wire2	1.093e-4	1.162e-4	6.28%
Wire3	5.864e-6	5.804e-6	-1.03%
Wire4	1.581e-5	1.543e-5	-2.39%
Wire5	1.058e-5	1.080e-5	2.11%
Wire6	1.019e-5	1.008e-5	-1.08%
Wire7	1.221e-4	1.140e-4	-6.71%
Wire8	3.750e-5	3.725e-5	-0.66%

4.5.5 Simulation efficiency study

As the model is trained on synthesized layout tiles, the trained model can be used to find the electric potential distribution for any layout with the discussed method, as long as the layout is synthesized with the same technology.

The runtime of the proposed method is tested with the batch size set to 100, on average it takes 34ms to complete one inference, which means 34ms per simulation. In comparison, it takes COMSOL 2 seconds to finish one simulation, which means the proposed method achieves 58x speedup. Both times include data pre-processing and moving to or from GPU, but do not include the library set-up time.

Another major drawback of COMSOL is that the architecture of this software is so well designed that the potential of parallelism in multiple simulations is limited. Normally, one machine can only run one COMSOL simulation at a time as it is already multi-threaded. This parallelism is crucial for the applications discussed as the layout is partitioned into a lot of tiles so running simulations in parallel would be of great help. The proposed method, on the other hand, can easily handle this level of parallelism by encoding simulation

problems into batches. This can further increase the advantage of the proposed method over COMSOL as all previous tests are done with batch size set to 1. Although it would take about 2 seconds to load the Tensorflow libraries and the saved neural network, this time can be neglected or amortized if there is a large number of problems to be solved, again because of the large number of tiles generated from layout partition.

Following this idea, the batch size is set to 100 for the model and the time to calculate electrostatics and TDDB lifetime for the complete M3 shown in Fig. 4.4 is compared. It takes 175 seconds with the proposed method. However, it takes COMSOL 6.7 hours. This means 138x speed up. Although COMSOL supports larger tiles, it takes longer to compute each tile and the total time would not reduce significantly.

4.5.6 Discussion of network structure

Besides the proposed network, two other network structures have also been trained in the experiment. The first one is a simple CNN based network, however without the max pooling or resize-convolutional layers for down/up sampling. Without down sampling, the resulting large feature maps greatly increase FLOPs (floating point operations), and thus slow down both training and inference. Consequently, the depth of the network is chosen to be 8, same as the decoder part of the proposed network. The other network tested is similar to the proposed network, except that all skip connections are removed.

Two metrics are used when testing the networks: accuracy of electric potential and TDDB aging analysis. The results are listed in Table 4.2, which are all displayed by difference in percentage against results from COMSOL, as in the tables above. The results of TDDB aging are from the same 8 wires used in the previous experiment. It can be seen

Table 4.2: Result discrepancy achieved with different network structures

	Potential	TDDB analysis
Proposed	1.0%	2.6%
Simple CNN	31.4%	73.4%
No-skip	15.4%	18.4%

that the proposed structure achieves the best result. For the simple CNN based network, it is greatly under-fitting and thus results in poor accuracy, which proves the necessity of a deeper network. Furthermore, it proves that the proposed network structure is able to well encode the electrostatics problem, and use the encoded information to derive the potential distribution. For the other network without skip connections, it starts to show over-fitting after 50 epochs. Eventually after 200 epochs, the training loss drops to 5%. However, the loss on test set is 15.4%, much higher than the proposed network. This proves that the skip connections are of key importance in solving the electrostatics problems.

4.6 Summary

This work proposes a method to use machine learning, specifically a CNN based neural network to solve electrostatics problems. The proposed method first encodes the electrostatic problem to be solved into an image. Taking it as input, the neural network then solves the problem through inference and outputs an image of electric potential distribution. Training and testing are done on a dataset from a synthesized CPU chip. Once trained, the model is applicable to any synthesized layout of the same technology. Compared to the conventional FEM based solver, the proposed method achieves 138x speedup, while keeping 99% of the accuracy on potential analysis, and 97% for TDDB aging analysis.

Chapter 5

GLU3.0: Fast GPU-based Parallel Sparse LU Factorization Solver

LU factorization of sparse matrices plays an important role in many engineering and scientific computing problems such as circuit simulation. But parallelizing LU factorization with the Graphic Processing Units (GPU) still remains a challenging problem due to high data dependency and irregular memory accesses. Recently the GPU-based hybrid right-looking sparse LU solver, called GLU (1.0 and 2.0), has been proposed to exploit the fine grain level parallelism of GPU. However, a new type of data dependency (called double-U dependency) introduced by GLU slows down the preprocessing step. Furthermore, GLU uses fixed GPU thread allocation strategy, which limits the parallelism. This chapter introduces an updated sparse LU factorization method, called *GLU3.0*, which solves the aforementioned problems. First, it introduces a much more efficient data dependency detection algorithm. Second, it is observed that the potential parallelism is different as

the matrix factorization goes on. Based on this, three different modes of GPU kernel are developed which adapt to different stages to accommodate the computing task changes in the factorization.

Experimental results on circuit matrices from University of Florida Sparse Matrix Collection (UFL) show that GLU3.0 delivers 2-3 orders of magnitude speedup over GLU2.0 for the data dependency detection. Furthermore, GLU3.0 achieve 13.0 × (arithmetic mean) or 6.7× (geometric mean) speedup over GLU2.0 and 7.1× (arithmetic mean) or 4.8 × (geometric mean) over the recently proposed enhanced GLU2.0 sparse LU solver on the same set of circuit matrices. This work is published in [52].

5.1 Overview and related works

A new version of GPU-based sparse LU factorization solver, called GLU3.0, is introduced in this chapter. It is based on existing GLU1.0/2.0 using hybrid right-looking LU factorization algorithm. The main improvements of GLU3.0 are summarized as follows:

- To mitigate the slow process to detect the new double-U data dependency in existing GLU2.0, GLU3.0 introduces a new dependency detection algorithm. It uses a relaxed principal to find all required dependencies, plus some redundant ones. The efficiency is a lot higher than the previous solution with little impact on performance.
- Based on the circuit matrices analyzed, a pattern of potential parallelism is observed, as the matrix factorization goes on. Basically, the number of columns and its associated subcolumns (updates) of each column, which is an important unit of parallel computing tasks, are inversely correlated. As a result, the number of columns can be

used as a good metric for resource allocation. Three different modes of GPU kernel have been developed that adapt to different stages to accommodate the computing task changes. As a result, GLU3.0 can dynamically allocate GPU threads and memory based on the number of columns in a level to better balance the computing demands and resources during the LU factorization process.

Numerical results on circuit matrices from University of Florida Sparse Matrix Collection (UFL) show that the GLU3.0 can deliver 2-3 orders of magnitude speedup over GLU2.0 for the data dependency detection. Furthermore, GLU3.0 *consistently* outperforms both GLU 2.0 and the recently proposed enhanced GLU2.0 sparse LU solver on the same set of circuit matrices. Furthermore, GLU3.0 achieve 13.0 $\times$ (arithmetic mean) or 6.7 $\times$ (geometric mean) speedup over GLU 2.0 and 7.1 $\times$ (arithmetic mean) or 4.8 $\times$ (geometric mean) over recent proposed enhanced GLU2.0 on the same set of circuit matrices.

5.2 Review of LU factorization and CUDA

This section briefly reviews the traditional G/P left-looking method for sparse matrices LU factorization [27] and the recently proposed hybrid right-looking algorithm used in GLU1.0, GLU2.0 and a recent GLU enhancement work [39]. The GPU architectures and NVIDIA CUDA programming system is also reviewed.

An example matrix is used for illustrating important concepts and algorithms in the following discussions. The matrix is shown in Fig. 5.1, where the colored spots stand for nonzero elements.

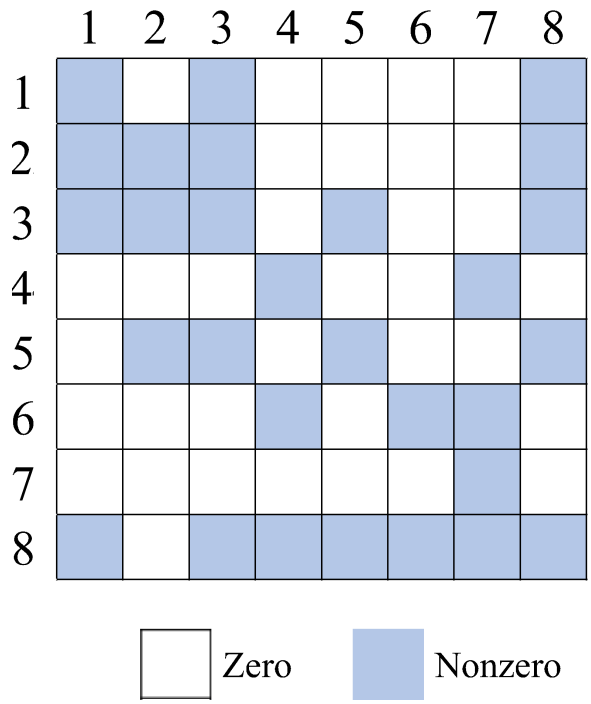


Figure 5.1: The example matrix

5.2.1 The left-looking method

Traditional Gaussian elimination based LU factorization method (also called right-looking method) solves one row for U matrix and then one column for L matrix in each iteration. While the G/P left-looking method computes one column in one iteration for both L and U instead, which is achieved by solving a lower triangular matrix. It also allows the symbolic fill-in analysis of L and U matrices before the actual numerical computing. As a result, G/P left-looking method shows better performance for sparse matrices, especially those from circuit simulations [19].

Algorithm 3 lists the detailed implementation of G/P left-looking LU factorization [27]. The input of this algorithm A_s is the nonzero filled-in matrix of A after symbolic analysis. The final result is $A_s = L + U - I$, where I is identity matrix. The matrix A_s

is factorized column by column (the outer j loop), and factorizing each column for both L and U contains two steps. The first step (lines 4-9) is to solve a triangular matrix. In each k loop, element-wise multiply-and-accumulate (MAC) operation is done (line 6-8) for the partial column vector $A_s(k+1:n, j)$. $A_s(i, k)$ are the elements in the factorized columns on the left of current column j. This is the reason why it is named left-looking LU method. Then the second step (lines 10-13) is a much simpler loop that finishes the factorization of this column. Triangular matrix solving (lines 4-9) is the most essential and computationally expensive step in this algorithm.

Algorithm 3 The Gilbert-Peierls left-looking algorithm

1: /* Scan each column from left to right */
2: **for** $j = 1$ to n **do**
3: /*Triangular matrix solving */
4: **for** $k = 1$ to $j - 1$ where $A_s(k, j) \neq 0$ **do**
5: /*Vector multiple-and-add (MAC) */
6: **for** $i = k + 1$ to n where $A_s(i, k) \neq 0$ **do**
7: $A_s(i, j) = A_s(i, j) - A_s(i, k) * A_s(k, j)$
8: **end for**
9: **end for**
10: /*Compute column j for L matrix*/
11: **for** $i = j + 1$ to n where $A_s(i, j) \neq 0$ **do**
12: $A_s(i, j) = A_s(i, j)/A_s(j, j)$
13: **end for**
14: **end for**

Fig. 5.2 gives a complete example of this step. In this example, column 7 is being factorized, meaning $j = 7$ in Algorithm 3. Only two k's satisfy $A_s(k, j) \neq 0$ (line 4), which are 4 and 6 (as $A_s(4, 7) \neq 0$ and $A_s(6, 7) \neq 0$). The two sub-figures show these two iterations respectively. In (a), $k = 4$, so column 4 is used to update column 7. The update operation refers to lines 6-8 of Algorithm 3, where two elements of column 7 ($A_s(6, 7)$ and $A_s(8, 7)$) are updated by MAC operations with the red elements in column 4 multiplying $A_s(4, 7)$.

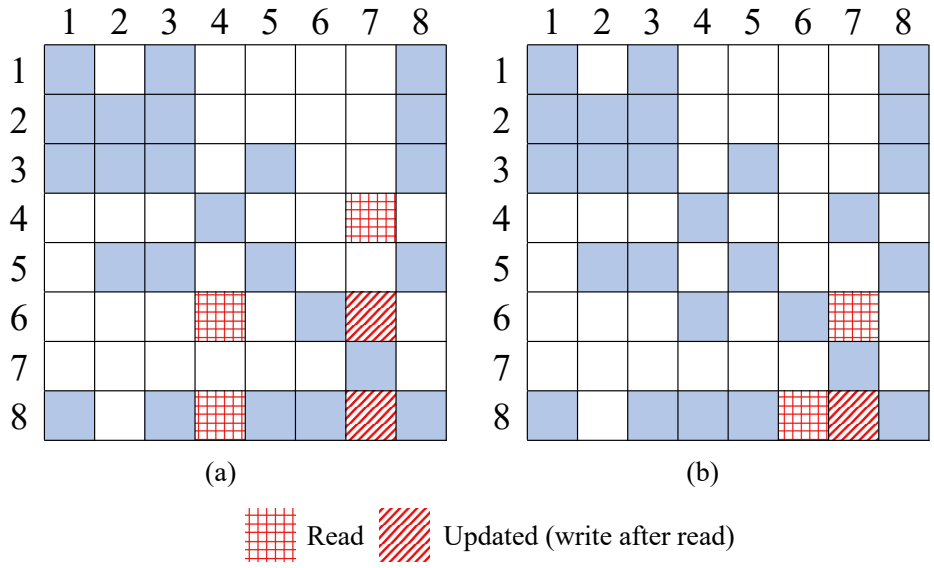


Figure 5.2: The two update iterations completing factorization of the 7th column ($j = 7$) (a) update using the 4th column ($k = 4$) (b) update using the 6th column ($k = 6$)

(b) shows the next iteration, where $k = 6$, column 3 is used to further update column 7, which can be explicitly written as $A_s(8,7) \leftarrow A_s(8,7) - A_s(8,6) * A_s(6,7)$.

5.2.2 The column-based right-looking method used in GLU

As elaborated in [30], the G/P left-looking sparse LU factorization has one limitation that it fails to parallelize the two loops in triangular matrix solving process (lines 4-8 of Algorithm 3). It can only work on (write) one column (current column j) at a time as indicated in line 7. To mitigate this problem, He *et al.* proposed the hybrid column-based right-looking LU factorization algorithm for GLU [30]. The algorithm is hybrid because it still keeps the column-based parallelism in the left-looking algorithm while updates columns on the right during factorization. Similar symbolic analysis is still applied in advance as well.

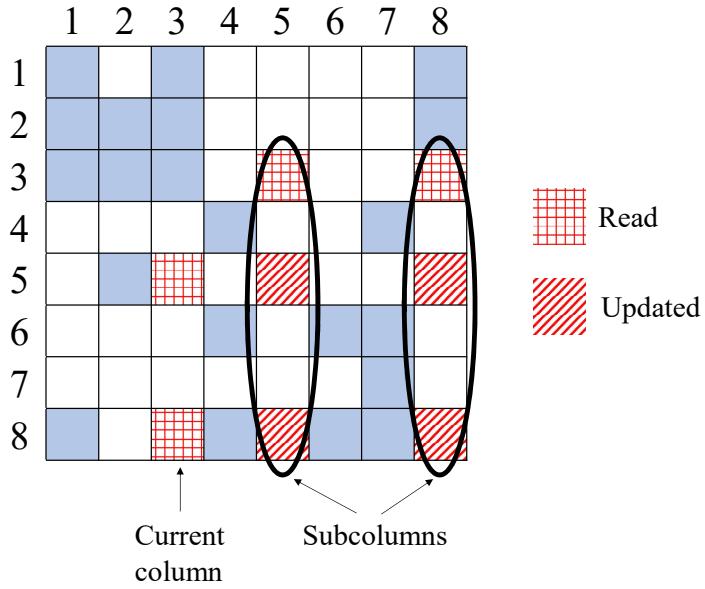


Figure 5.3: Subcolumns and submatrix column 3. All highlighted elements compose the submatrix, which include elements being read and elements being updated.

The hybrid right-looking LU factorization algorithm is listed in Algorithm 4. Similarly, the current column under computing is indexed by j. For each column, the first step is to compute the L part of the current column (lines 4-6), which is equivalent to lines 10-12 of Algorithm 3. Then, it looks right to find all columns k ($k > j$) that meet $A_s(k, j) \neq 0$, and uses the currently factorized column j to update these columns (lines 8-12). For the sake of presentation convenience without confusion, these columns are named *subcolumns* of column j. Note that these subcolumns are not part of the column j. Furthermore, this step of updating all subcolumns is called *submatrix update*, where all elements being read or updated form a *submatrix*. Fig. 5.3 gives an example illustrating these two concepts. In the figure, $j = 3$, its subcolumns are column 5 and 8, because $A_s(3, 5)$ and $A_s(3, 8)$ are nonzero elements. Corresponding this to the execution of Algorithm 4, during iteration $j = 3$, two k's meet the condition of line 8, which are 5 and 8.

Algorithm 4 The hybrid column-based right-looking algorithm for GLU1.0/2.0

```
1: /* Scan each column from left to right */
2: for j = 1 to n do
3:    /*Compute column j of L matrix*/
4:    for k = j + 1 to n where A_s(k, j) ≠ 0 do
5:       A_s(k, j) = A_s(k, j)/A_s(j, j)
6:    end for
7:    /*Update the submatrix for next iteration*/
8:    for k = j + 1 to n where A_s(j, k) ≠ 0 do
9:       for i = j + 1 to n where A_s(i, j) ≠ 0 do
10:         A_s(i, k) = A_s(i, k) − A_s(i, j) ∗ A_s(j, k)
11:      end for
12:   end for
13: end for
```

The key difference between this right-looking algorithm and the left-looking one is that submatrix update completes the equivalent jobs of triangular matrix solving (lines 4-9 of Algorithm 3) in advance. In the example shown in Fig. 5.2, both update operations are completed while $j = 7$. However, in the case of the right-looking algorithm, the update in (a) is done while $j = 4$, and update in (b) is done while $j = 6$. As will be discussed in detail in the following section, this difference enables exploiting parallelization between subcolumns.

5.2.3 Additional data dependency in GLU: the fix in GLU2.0

Data dependency is an important issue in parallel computing or general high performance computing. It puts hard requirements in the orders of operations. In SuperLU [22] and NICSLU [12], elimination tree has been used to resolve this issue.

For GLU, in order to factorize several columns in parallel, data dependency between columns needs to be detected in the first place. With complete information of dependency, columns can be grouped into *levels*, where all columns in the same level are independent of each other and can thus be factorized in parallel. Such process deriving information about levels is called *levelization*, which is a similar method to elimination tree. In the left-looking LU factorization method, levelization is done by studying the sparsity pattern of the U matrix. Any $U(i, j) \neq 0, i < j$ results in column j being dependent on i because of the triangular matrix solving (lines 4-9 in Algorithm 3). This dependency detection algorithm was also used in GLU1.0. Algorithm 5 lists the complete flow of this.

Algorithm 5 Column dependency detection algorithm used in GLU1.0

```
1: for i = 1 to n do
2:    /* Look up for all nonzeros in column i of U */
3:    for j = 1 to i − 1 where A_s(j, i) ≠ 0 do
4:       if Column j of L is not empty then
5:          Add j to i's dependency list
6:       end if
7:    end for
8: end for
```

However, as reported in GLU2.0 and [39], the hybrid right-looking algorithm used in GLU leads to a new column dependency named *double-U dependency*, originating from the read-write hazard during parallel submatrix updates. An example of this can be found in columns 4 and 6 of the example matrix, with the details highlighted in Fig. 5.4. In (a), $A_s(6,7)$ is updated by column 4: $A_s(6,7) \leftarrow A_s(6,7) - A_s(6,4) * A_s(4,7)$. In (b), $A_s(6,7)$ is used to update column 7: $A_s(8,7) \leftarrow A_s(8,7) - A_s(8,6) * A_s(6,7)$. In the scheme of GLU1.0, both updates are executed in parallel. However, $A_s(6,7)$ is written in (a) and read in (b), which forms a read-write hazard when they are executed in parallel. To ensure correctness, the write operation in (a) must finish before the read operation in (b). As a result, an additional dependency between columns 4 and 6 needs to be introduced undesirably.

Such read-write dependency is called double-U dependency in GLU2.0 as it originates from two overlapped U-shaped dependencies as shown in Fig. 5.4. To detect this new dependency, GLU2.0 introduced a different dependency detection process as shown in

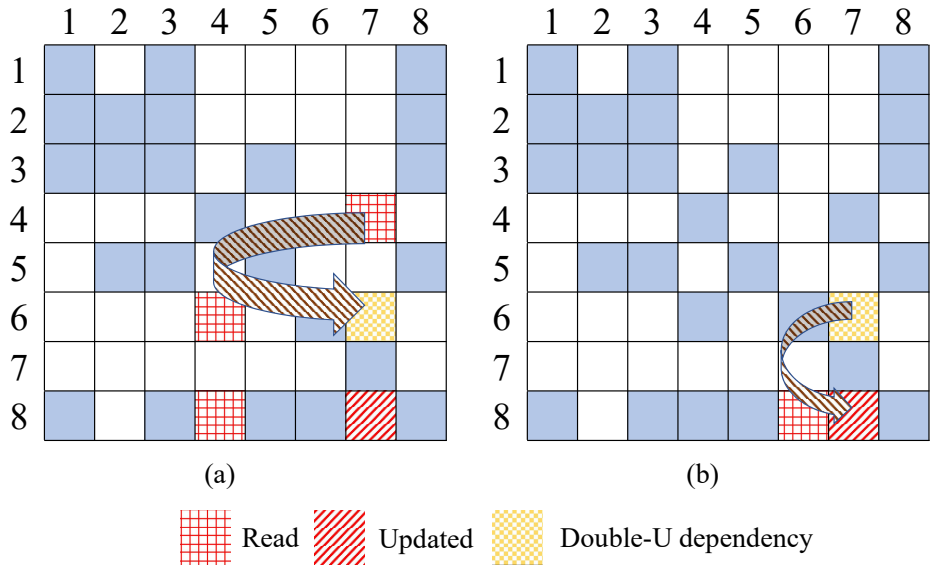


Figure 5.4: An example of double-U dependency originated from element (6,7)

Algorithm 6. This algorithm directly looks for double-U dependency. Suppose k is found for given i, t and j, $A_s(t,k)$ is updated by $A_s(t,i)$, while it is also used to update $A_s(j,k)$. As a result a double-U dependency exists between columns i and t. In the example of Fig. 5.4, $i = 4, t = 6, j = 8$, and $k = 7$ respectively.

However, this detection algorithm can be quite expensive because of the three nested loops that have $O(n^3)$ complexity. In comparison, there are only two for loops in the U matrix pattern based dependency detection algorithm. It leads to performance degradation compared to GLU1.0.

Besides dependency detection and levelization, some preprocessing and symbolic analysis needs to be done on CPU ahead of factorization. The preprocessing includes MC64 and AMD (Approximate minimum degree) algorithms in order to reduce the number

Algorithm 6 Double-U dependency detection algorithm used in GLU2.0

```
1: for i = 1 to n do
2:     Store all non-zero indices of row i in I_i
3:     for t = i to n where A_s(t, i) ≠ 0 do
4:         for j = t to n where A_s(j, t) ≠ 0 do
5:             Store all non-zero indices of row j in I_j
6:             if ∃k, k ∈ I_i, k ∈ I_j, k > t then
7:                 Add i to t's dependency list
8:             end if
9:         end for
10:    end for
11: end for
```

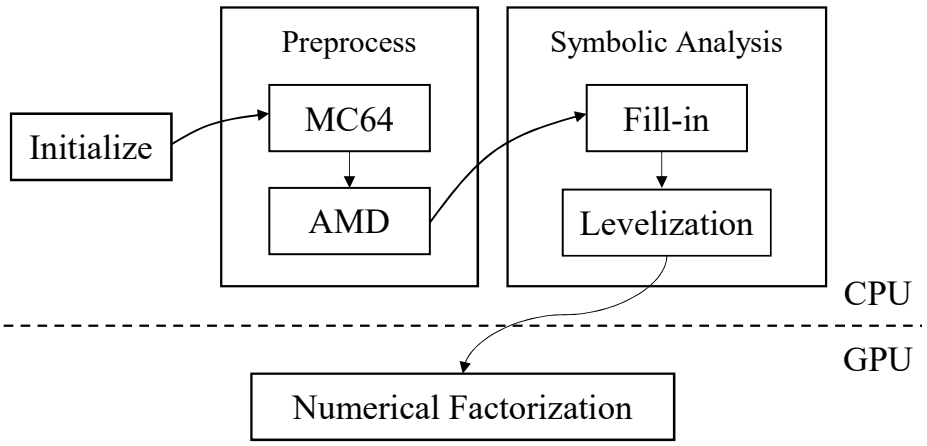


Figure 5.5: Complete flow of GLU2.0

of final nonzero elements, as is done in NICSLU [12]. Symbolic analysis includes fill-in and levelization. Combining all this, we have the complete flow of GLU2.0 shown in Fig. 5.5.

5.2.4 Enhancements to GLU2.0

Recently, Lee *et. al* proposed a method to enhance GLU2.0[39]. In detail, three different kernels were proposed, namely cluster mode, batch mode and pipeline mode. Modes are selected based on different number of columns in levels. In batch mode and pipeline mode, overlapped execution of different levels is achieved to some extent, which contributes to the speed-up. Besides this, kernel launches are managed by a small kernel instead of CPU, which is called dynamic parallelism enabled by CUDA compute capability 3.0. Combining these techniques, the enhanced GLU has achieved 1.26× (geometric mean) speedup over GLU2.0.

5.2.5 Review of GPU Architecture and CUDA programming

CUDA, short for Compute Unified Device Architecture, is the parallel programming model for NVIDIA's general-purpose GPUs. The architecture of a typical CUDA-capable GPU is consisted of an array of highly threaded streaming multiprocessors (SM) and comes with a huge amount of DRAM, referred to as global memory. Take the GTX TITAN X GPU for example. It contains 24 SMs, each of which has 128 streaming multiprocessors (SPs, or CUDA cores called by NVIDIA), 8 special function units (SFU), and its own shared memory/L1 cache. The architecture of the GPU and streaming multiprocessors is shown in Fig. 5.6.

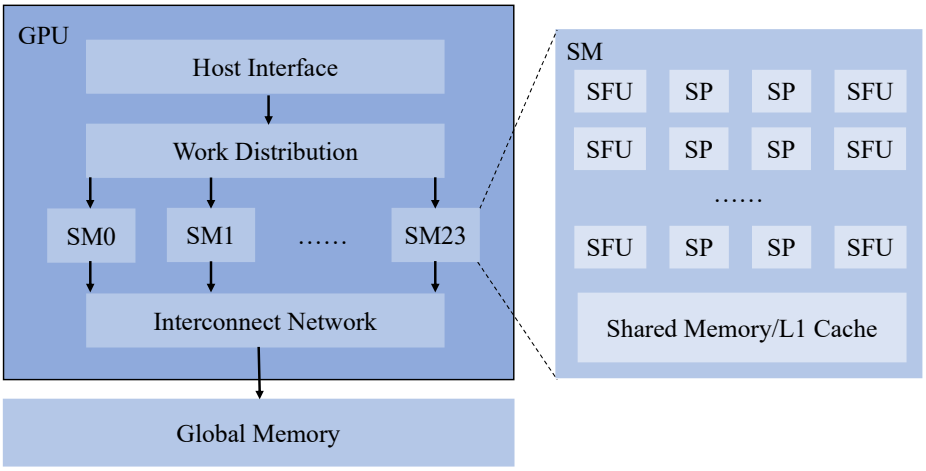


Figure 5.6: Diagram of NVIDIA TITAN X and the streaming multiprocessor. (SP is short for streaming processor, L/S for load/store unit, and SFU for Special Function Unit.)

As the programming model of GPU, CUDA extends C into CUDA C and supports such tasks as threads calling and memory allocation, which makes programmers able to explore most of the capabilities of GPU parallelism. In CUDA programming model, illustrated in Fig. 5.7, threads are organized into blocks; blocks of threads are organized as grids. CUDA also assumes that both the host (CPU) and the device (GPU) maintain their own separate memory spaces, which are referred to as host memory and device memory respectively. For every block of threads, a shared memory is accessible to all threads in that same block. The global memory is accessible to all threads in all blocks. Developers can write programs running millions of threads with thousands of blocks in parallel. This massive parallelism forms the reason that programs with GPU acceleration can be much faster than their CPU counterparts. CUDA C provides its extended keywords and built-in variables, such as `blockIdx.{x,y,z}` and `threadIdx.{x,y.z}`, to assign unique ID to all blocks and threads in the whole grid partition. Therefore, programmers can easily map the data partition to the parallel threads, and instruct the specific thread to compute its own

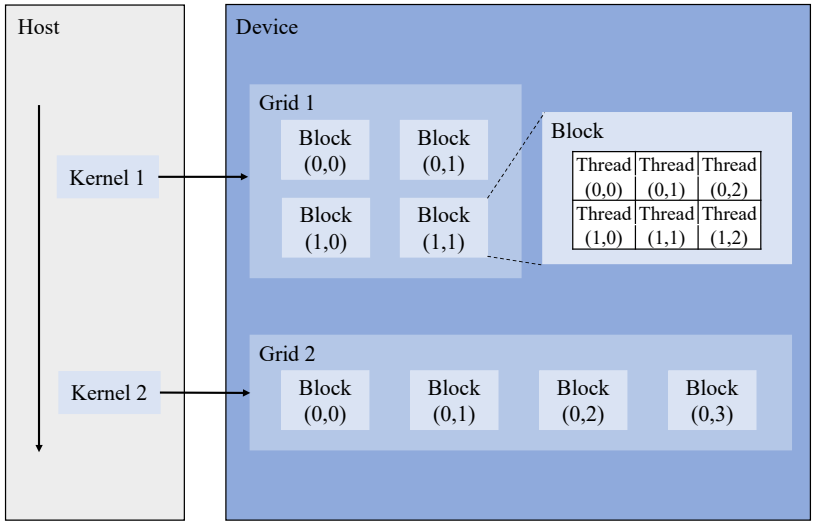


Figure 5.7: The programming model of CUDA.

responsible data elements. Fig. 5.7 shows an example of 2-dim blocks and 2-dim threads in a grid, the block ID and thread ID are indicated by their row and column positions.

5.3 New GPU based sparse LU solver: GLU 3.0

As introduced above, the work flow of factorizing a sparse matrix with GLU can be divided into two parts: the preprocessing and symbolic analysis on CPU and the numeric factorization on GPU. The second part on GPU might be repeated for many times when solving a nonlinear equation with Newton-Raphson method in circuit simulation applications. GLU3.0 significantly improves both the symbolic analysis and numeric factorization.

5.3.1 Relaxed data dependency detection method for GLU

As mentioned in Section 5.2.3, the prior dependency detection algorithm introduced to cover double-U dependency slows down the factorization a lot. This work solves

this problem with a better dependency detection algorithm, named *relaxed* column dependency detection method, which can reduce the process down to two loops. The new algorithm is based on the observation that a *necessary condition* for such additional dependency is the existence of nonzero elements on the left of diagonal element in the L matrix. In the example in Fig. 5.4, such dependency exists between columns 4 and 6. The nonzero element $A_s(6,4)$ on the left of diagonal element $A_s(6,6)$ is the necessary condition that column 6 depending on column 4, as it is the reason that $A_s(6,7)$ gets updated, and $A_s(6,7)$ is the very element that induces the double-U dependency.

Based on this observation, the new method simply just look for nonzero elements on the left of diagonal element, which can be called simply as "left looking", to find such new dependency. It is very similar to the "up looking" in the U matrix based dependency detection method employed in the left-looking factorization algorithm. Fig. 5.8 compares the result of them by applying both methods to column 6. As there is no nonzero element in column 6 of U matrix, "Looking up" from $A_s(6,6)$ will find no depended column of column 6. On the other hand, "looking left" from the same element, a nonzero element in column 4 can be seen, which is interpreted as the new dependency between columns 4 and 6 that is the double-U dependency as expected. The complete algorithm incorporating the new dependency detection method is listed in Algorithm 7. Lines 8-11 are the additional "left looking" part that is added to the original dependency detection algorithm listed in Algorithm 5.

In order to compare the aforementioned three dependency detection methods, they are applied to the example matrix from Fig. 5.2 and the results are shown in Fig. 5.9

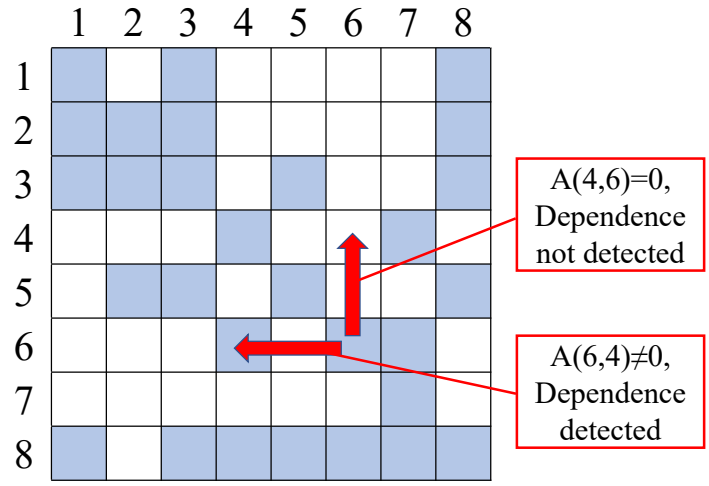


Figure 5.8: Comparison of left looking and up looking, left looking is able to detect double-U dependency.

respectively. An edge $x \rightarrow y$ indicates that that column x depends on column y. Comparing (a) and (b), the extra dependencies $1 \rightarrow 2$ and $4 \rightarrow 6$ (marked by blue line) are the double-U dependencies. Further comparing (b) and (c), one can see that the proposed method is able to detect all required column dependencies, plus a few redundant ones marked by red. Despite the redundant dependencies, the result of levelization is exactly the same, which means the same numerical performance on GPU can be expected. This example shows that the redundant dependencies have minor, if none, impacts on parallelism exploration of GLU. The reason why this dependency detection method is called *relaxed* is that it does not detect the exact set of dependencies, but a sufficient one possibly with some redundant dependencies. More examples about this will be reported later in Section 5.4.

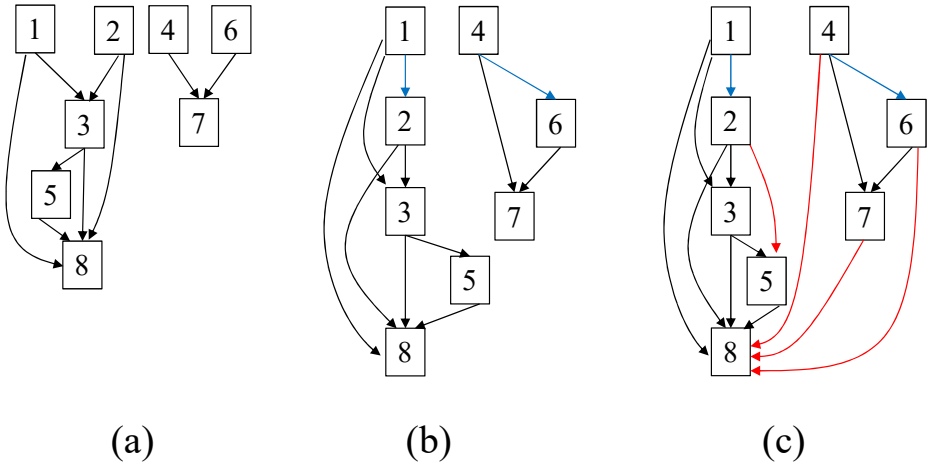


Figure 5.9: Dependency graph generated from 3 methods: (a) GLU1.0: incorrect result (b) GLU2.0: correct result (c) This work: the relaxed data dependency

Algorithm 7 The proposed relaxed column dependency detection method

1: **for** $k = 1$ to n **do**
2: /* Look up for all nonzeros in column k of U */
3: **for** $i = 1$ to $k - 1$ where $A_s(i, k) \neq 0$ **do**
4: **if** Column i of L is not empty **then**
5: Add i to k's dependency list
6: **end if**
7: **end for**
8: /* Look left for all nonzeros in row k of L */
9: **for** $i = 1$ to $k - 1$ where $A_s(k, i) \neq 0$ **do**
10: Add i to k's dependency list
11: **end for**
12: **end for**

5.3.2 New GPU kernels

Before going on to discuss the new GPU kernels, it is good to first review the submatrix update in GLU, which is a key step in Algorithm 4.

The submatrix update revisited

The submatrix update is explicitly listed in Algorithm 8 below. Specifically, we

Algorithm 8 The submatrix update in GLU

```
1: /*Update the submatrix for next iteration*/
2: for k = j + 1 to n where A_s(j, k) ≠ 0 do
3:     for i = j + 1 to n where A_s(i, j) ≠ 0 do
4:         A_s(i, k) = A_s(i, k) − A_s(i, j) ∗ A_s(j, k)
5:     end for
6: end for
```

can write the submatrix to be updated as

$$A_{sub} = \begin{bmatrix} A_s(j+1,j+1) & \cdots & A_s(j+1,n) \\ \vdots & \ddots & \vdots \\ A_s(n,j+1) & \cdots & A_s(n,n) \end{bmatrix} \tag{5.1}$$

where the size of the submatrix is $N \times N$, with $N = n - j$. The submatrix update operation can be further represented in the following format:

$$A_{sub} \leftarrow A_{sub} - \begin{bmatrix} A_s(j+1,j) \\ \vdots \\ A_s(n,j) \end{bmatrix} \cdot [A_s(j,j+1), \cdots, A_s(j,n)] \tag{5.2}$$

where the size of the two vectors are $N \times 1$, and $1 \times N$. Both two vectors and A_{sub} matrix are sparse. From (5.2), we can see that the submatrix update consists of two operations: (a) vector tensor multiplication (the second item on the right hand side); (b) matrix addition.

In the implementation of GLU, the submatrix update

$$A_{sub} - \begin{bmatrix} A_s(j+1,j) \\ \vdots \\ A_s(n,j) \end{bmatrix} \cdot [A_s(j,j+1), \cdots, A_s(j,n)]$$

is done in a column-wise way as depicted in (5.3):

$$\vec{A_s}(j+1:n,i) - \vec{A_s}(j+1:n,j) \cdot A_s(j,i), \text{for } i = [j+1, \cdots, n] \tag{5.3}$$

where

$$\vec{A_s}(j+1:n,i) = [A_s(j+1,i), \ldots, A_s(n,i)]^T$$

$$\vec{A_s}(j+1:n,j) = [A_s(j+1,j), \ldots, A_s(n,j)]^T.$$

As can be seen, the submatrix update consists of vector operations or *subcolumn update*. Each time, we can update one *subcolumn* i as shown in (5.3). This can be parallelized

in GPU where each resulting element can be computed using one thread, where the operation is multiply-accumulate (MAC) operation. There are two levels of parallelism: namely (a) the vector operations (or subcolumn updates) for different vectors as shown in (5.3) and (b) element-wise MAC operations in each vector or subcolumn. In contrast, the left-looking algorithm only has element-wise MAC operation parallelism in the triangular matrix solving process.

New adaptive GPU kernel

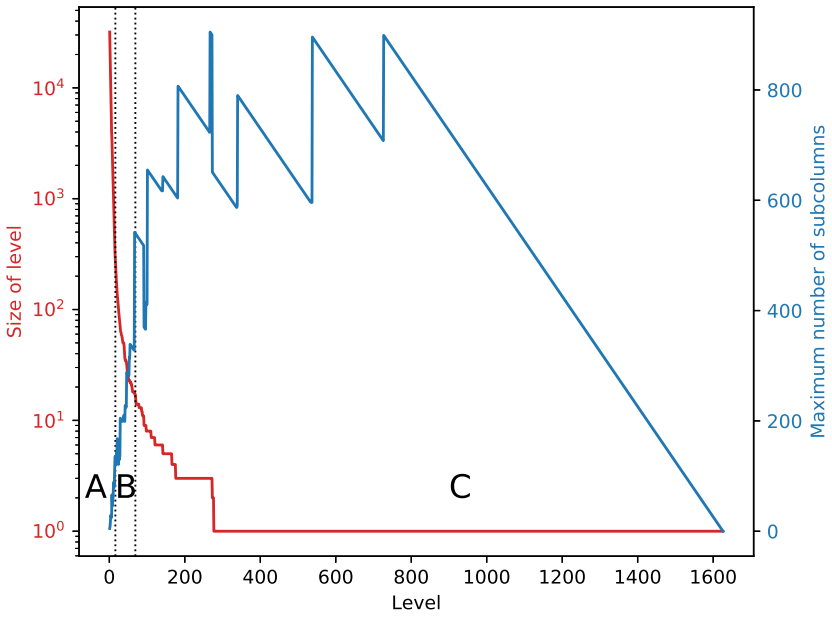
GLU1.0/2.0 used fixed resource allocation strategy in the GPU kernel. However, as the matrix size grows, the fixed resource allocation strategy will significantly restrict the potential parallelism in GPU.

Before going into details, it is helpful to first define several terms for the ease of discussion. All columns in the same level that can be factorized in parallel referred to as *parallelizable*; the *size* of a level is the number of parallelizable columns in this level. In other words, a *large* level has many parallelizable columns, while a *small* level has few columns.

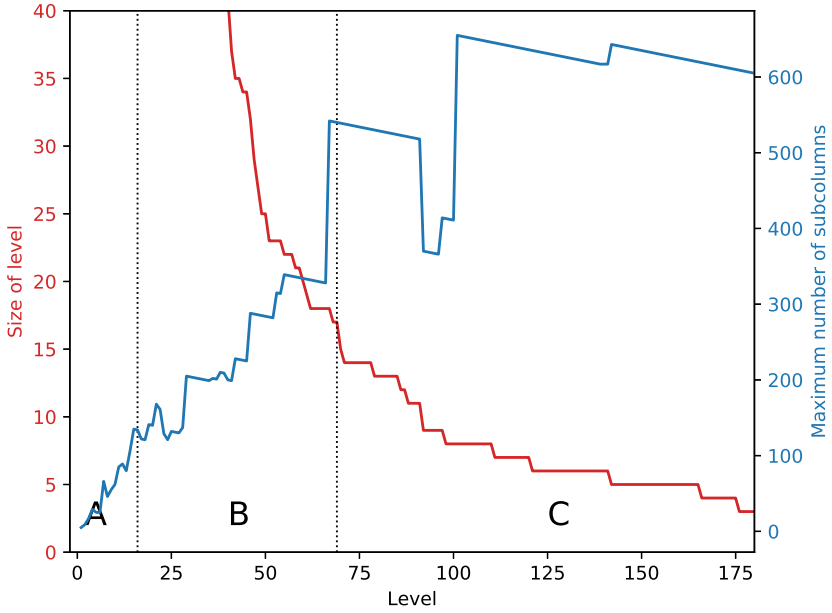
As defined, all columns in one level are parallelizable, and each column has many associated parallelizable subcolumns. This two-level parallelism distinguishes GLU from other parallel sparse matrix LU factorization algorithms. Two metrics can be used to describe the potential parallelism respectively, namely the size of one level, and the maximum number of subcolumns for all columns in one level, because they are the basic units that get parallelized. The first one is self-explanatory, and the second one is based on the fact that updating the submatrix with more subcolumns generally involves more calculations.

The potential parallelism keeps changing across the levels, which is the key reason of the fixed resource allocation strategy being inefficient. The trend of potential parallelism is shown in Fig. 5.10, which shows the size of each level and maximum number of subcolumns of the corresponding level from matrix ASIC100ks [21]. Note that although the number of subcolumns are different for each column in one level, the difference is very small. Consequently, it is easier to use the maximum number of subcolumns to represent that of one level. An important observation is that *levels generally fall into three categories*, which are also labeled in the figure. Type A levels in the beginning stage of factorization have huge number of parallelizable columns, while each column has very few associated subcolumns. For higher throughput, parallelizing columns should be prioritized for this type of levels. Type C levels, in contrast, have limited number of columns, while each column generally has large number of subcolumns until very end of the factorization process. As a result parallelizing subcolumns is more important for this type of levels. Type B levels, in the transitional stage, have great numbers of columns, and at the same time columns also have many subcolumns. So parallelism should be naturally balanced between them.

Furthermore, the second important observation is that the number of parallelizable columns and their associated subcolumns are inversely correlated in general. As a result, the size of a level can be used as a good estimation of the associated subcolumn numbers to dynamically allocate the computing resources to further improve the GPU kernel computing efficiency. Based on this observation, three computing modes of GLU kernels are developed, which are chosen based on the level sizes in a progressive way to accommodate the three types of levels.



(a) Level versus its size and the maximum number of subcolumns



(b) Zoomed in view

Figure 5.10: Number of columns and subcolumns of different levels

1. **Small block mode:** This mode is designed for type A levels. A convention followed from GLU for this mode is that one block takes care of a column, and one warp is assigned to a subcolumn. In this mode, as shown in Fig. 5.11(a), fewer warps are assigned to a CUDA block, which is why it is named small block mode. As the total number of warps is fixed for a given GPU, more blocks, or equivalently the factorization of more columns, can be carried out in parallel, which fits the requirement of type A levels: huge number of columns with a few subcolumns. Another important observation from Fig. 5.10 is that the number of subcolumns is gradually increasing, and the level size is decreasing quickly. In order to adapt to this change, the number of warps assigned to a block is gradually increased, assisting the growing number of subcolumns and trying to make full use of available warps at the same time. The number of warps assigned to a block grows from 2 to 4, 8, and eventually to 32, which is the number in the next mode. The exact number of warps assigned to a block is determined by number of columns in a level using following expression:

$$W = \frac{\text{Total number of warps}}{\text{Level size}} \tag{5.4}$$

where W is the number of warps assigned to each block.

Another factor limiting the number of possible parallel columns is memory. Because the columns being factorized are stored as a dense form in global memory, too many columns from a big level can overflow the memory. Specifically, during factorization of each column, an array of size n is allocated for caching. As a result, the maximum parallelizable columns N can be calculated as:

$$N = \frac{\text{Max global memory allowed}}{n * \text{sizeof(float)}} \tag{5.5}$$

where n is the number of rows of the matrix.

2. **Large block mode:** This mode takes care of type B levels, and it is similar to the kernel used in previous GLU versions. Same as the small block mode, each block takes care of one column and each warp is assigned to a subcolumn. In this mode, the number of subcolumns still keep growing, the number of threads each subcolumn gets (32, one warp) becomes insufficient. However, the maximum number of a thread block (1024) prohibits any further increase in this number.
3. **Stream mode:** To tackle maximum warp size (32) problem, Stream mode is proposed for type C levels in this work. In this mode, blocks instead of warps are assigned to each subcolumn, and therefore kernel calls instead of blocks are assigned to each column, and a block is assigned to each subcolumn now, as is shown in Fig. 5.11(c). To fully exploit parallelism within the same level, CUDAStreams are used, which allows parallel kernel execution through streams in a GPU. Although the number of CUDAStreams could also be dynamically, it has been observed that creating more CUDAStreams sometimes has a negative effect in performance. As a result, the number of CUDAStreams has been set to a fixed number, 16. This number is able to produce optimal results based on the experimental results which will be discussed later. Accordingly, this mode begins as long as the level size drops to 16.

Remark that the three GPU kernel modes proposed are quite different than then three modes proposed in [39]. First, the proposed approach is based on the observation of both parallelizable column count and associated subcolumn count change with different levels, while Lee's work is only based on the column count in each level. Second, the

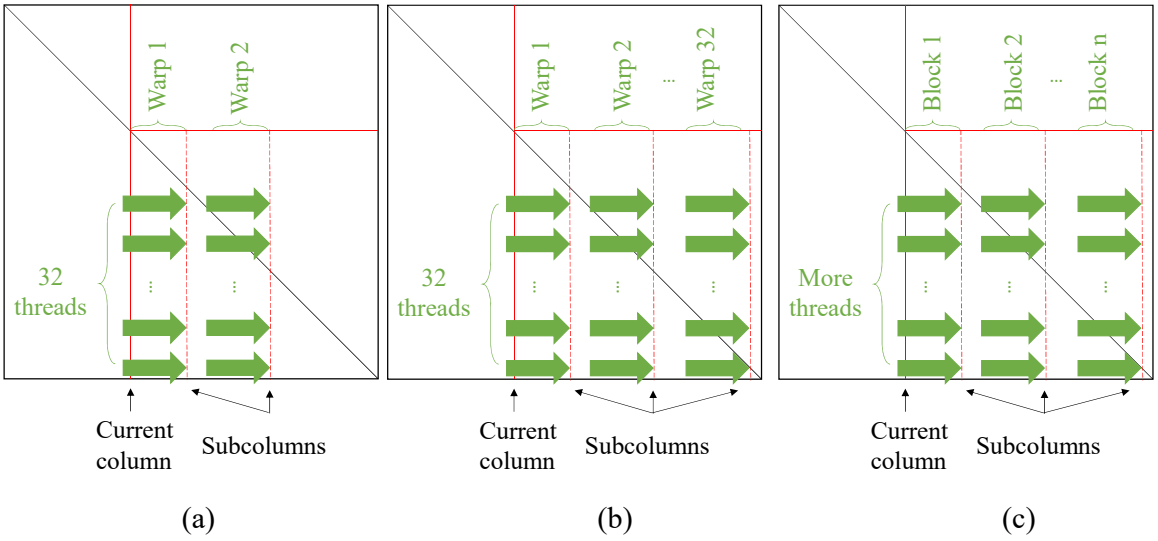


Figure 5.11: Comparison of the concurrency layout for one column in different kernels: (a) Small block mode (b) Large block mode (c) Stream mode

proposed approach dynamically allocates GPU computing resources (different number of warps and threads in each blocks etc) based on those information, while Lee's work exploits some advanced GPU features such dynamic kernel launch. Third, Lee's work focus more on exploit parallelism between different levels, while GLU 3.0 focus on dynamically changing parallelisms in one level over the course of factorization process.

5.4 Numerical results and discussions

The proposed GLU3.0 is implemented in C++ and CUDA-C, and compiled with optimization level 3 (-O3). The tests were run on a server equipped with Intel(R) Xeon(R) CPU E5-2698 v3, 128GB RAM, and NVIDIA GTX TITAN X (3072 CUDA cores, 12GB GDDR5 memory, Maxwell architecture). The test matrices come from the University of Florida Sparse Matrix Collection [21], and the same ones tested in [39] are used for the sake of comparison. Single precision floating point is used for computation as the Maxwell

architecture does not support atomic operations for double precision. On newer GPU platforms that allow double atomic operations, the performance of GLU with double precision is expected on average 30% slower compared to that of the single precision version [39].

First the new relaxed data dependency detection method proposed in Section 5.3.1 is tested. It is applied to the test matrices to perform levelization. The results of levelization are presented in Table 5.2. More details of test matrices such as number of non-zeros can be found in Table 5.1.

Two observations can be drawn from this table. First, the number of additional levels resulting from the new dependency detection method are just a few or even zero. As the number of levels is the most decisive parameter of runtime of the GPU kernel, this means the proposed new leveling algorithm would have marginal impacts on the runtime of numerical factorization on GPU. Second, the runtime of levelization (Algorithm 7) has improved dramatically on all test matrices compared to the existing method. The previous method of levelization used in GLU2.0 (Algorithm 6) has to explicitly find all double-U dependency, which has $O(n^3)$ complexity and thus makes the runtime of preprocessing non-negligible (compared to the LU factorization time). However, with an average speed-up ratio of 8804.1 (arithmetic average) or 3145.8 (geometric mean), the proposed new method is able to reduce the preprocessing runtime back into the similar time frame of the preprocessing time in the plain left-looking based method.

Then the performance of GPU kernels of GLU3.0 is tested and compared with GLU2.0 and NICSLU [12]. 32 threads are used when testing the performance of NICSLU. The results are presented in Table 5.1. The speed-up ratio of the proposed work over [39]

Table 5.1: Solver runtimes of GLU3.0 vs previous works, where *nz* stands for number of nonzeros before fill-in, and *nnz* stands for number of nonzeros after fill-in

Matrix	Number of rows	nz	nnz	CPU time (ms)		Numerical factorization time (ms)					
				GLU3.0	GLU2.0	GLU3.0 (GPU)	GLU2.0 (GPU)	NICSLU (CPU) [12]	Speed-up over GLU2.0	Speed-up over [39]	Speed-up over [12]
rajat12	1879	12926	13948	3.999	13.998	2.237	2.44883	3.99	1.1	1.0	1.78
circuit_2	4510	21199	32671	7.998	59.991	4.144	8.36301	6.66	2.0	1.9	1.61
memplus	17758	126150	126152	15.997	377.943	6.672	6.90432	26.91	1.0	0.9	4.03
rajat27	20640	99777	143438	21.997	404.939	10.539	23.8673	34.44	2.3	2.0	3.27
onetone2	36057	227628	1306245	353.946	36729.4	60.964	550.598	432.69	9.0	8.3	7.10
rajat15	37261	443573	1697198	423.936	18461.2	71.135	458.611	356.90	6.4	6.1	5.02
rajat26	51032	249302	343497	76.988	2011.69	32.366	104.12	88.77	3.2	4.2	2.74
circuit_4	80209	307604	438628	295.955	4662.29	68.944	394.995	118.23	5.7	9.1	1.71
rajat20	86916	605045	2204552	2190.67	121207	241.822	2538.24	245.63	10.5	8.8	1.02
ASIC_100ks	99190	578890	3638758	2052.69	316998	215.493	2652.79	357.53	12.3	14.1	1.66
hcircuit	105676	513072	630666	67.99	6279.05	46.996	243.846	221.50	5.2	9.5	4.71
Raj1	263743	1302464	7287722	7240.9	140008	845.189	7969.05	825.38	9.4	8.7	0.98
ASIC_320ks	321671	1827807	4838825	2336.64	410679	216.517	5632.8	765.35	26.0	21.3	3.53
ASIC_680ks	682712	2329176	4957172	1747.73	686421	210.697	11771.7	614.75	55.9	18.4	2.92
G3_circuit	1585478	4623152	36699336	9728.52	1764580	878.153	38780.9	9232.618	44.2	8.2	10.51
								Arithmetic mean	13.0	7.1	3.51
								Geometric mean	6.7	4.8	2.81

Table 5.2: Levelization runtimes

Matrix	Number of levels		Levelization Time (ms)		
	GLU2.0	this work	GLU2.0	this work	speed-up
rajat12	37	39	3.048	0.035	87.1
circuit_2	101	102	17.187	0.074	232.3
memplus	147	147	345.568	0.234	1476.8
rajat27	123	125	272.216	0.32	850.7
onetone2	1213	1213	4009.51	1.589	2523.3
rajat15	968	968	3680.02	2.224	1654.7
rajat26	157	158	1703.92	0.711	2396.5
circuit_4	228	229	5053.39	0.944	5353.2
rajat20	1216	1219	15931.2	3.389	4700.9
ASIC_100ks	1626	1626	36388.8	5.301	6864.5
hcircuit	144	145	6122.57	1.206	5076.8
Raj1	1594	1595	56580.9	11.102	5096.5
ASIC_320ks	1669	1669	168979	8.573	19710.6
ASIC_680ks	1450	1450	530478	10.642	49847.6
G3_circuit	652	688	1741860	66.508	26190.2
				Arithmetic mean	8804.1
				Geometric mean	3145.8

is calculated based on its reported speed-up ratio against GLU2.0 using the same testing matrices. The runtime measured includes the time completing memory copy. CPU time that comprises of preprocessing and symbolic analysis is compared as well. As can be seen from the table, despite slightly more levels as reported in Table 5.2, the proposed new GPU kernel still demonstrates a steady speedup over the kernels from GLU2.0 and the improved version from [39]. At least 5x speed-up can be achieved on average. Furthermore, more significant improvement can be expected when it comes to bigger matrices, starting from circuit_4 with a row number of 80209. The reason is that the computational tasks of small matrices are so light that the GPU still allows more parallelizable tasks. On the other hand, when factorizing larger matrices, the limited GPU computation power will throttle

Table 5.3: GPU kernel runtimes without enabling all 3 kernel modes, compared to case 1 where small block mode is disabled, and case 2 where stream mode is disabled.

Matrix	GPU time (ms)			Level distribution		
	GLU3.0	Case 1	Case 2	A	B	C
rajat12	2.237	2.776	2.158	2	4	33
circuit_2	4.144	4.871	4.650	1	10	91
memplus	6.672	9.364	7.187	4	3	140
rajat27	10.539	13.069	10.665	6	23	96
onetone2	60.964	66.126	173.863	14	33	1166
rajat15	71.135	82.677	163.947	11	96	861
rajat26	32.366	43.697	35.330	8	36	114
circuit_4	68.944	170.49	103.515	7	9	213
rajat20	241.822	571.95	1019.12	11	41	1167
ASIC_100ks	215.493	246.84	1047.78	13	56	1557
hcircuit	46.996	59.103	47.761	10	14	121
Raj1	845.189	2611.12	2115	29	223	1343
ASIC_320ks	216.517	311.778	1094.78	14	50	1605
ASIC_680ks	210.697	279.784	721.589	14	55	1381
G3_circuit	878.153	783.592	877.444	104	327	257

full parallelization in the GLU. In these cases, the proposed adaptive kernels can utilize the GPU in a better way so that more parallelism and shorter runtime is achieved.

To further validate the improvement from the proposed three modes of kernels, another experiment is conducted, where either one of the two newly proposed modes (small block mode and stream mode) are disabled, to show the degradation of performance without them. The results are listed in Table 5.3. In case 1, small mode is disabled. While in case 2, stream mode is disabled. The number of three different types of levels are also listed. Comparing GLU3.0 with case 1, we can see that small block mode benefits most matrices except G3_circuit. Although the number of type A levels is generally small, small block mode can still lead to decent improvement. The reason of G3_circuit being slower without small block mode is probably that the number of blocks assigned in small block mode is less

than optimal because the limitation of (5.5). In this case, more warps should be assigned to a block as the total number of blocks is limited. Then comparing GLU3.0 with case 2, a more significant improvement can be seen from stream mode. Furthermore, stream mode tend to benefit all matrices, as the results of GLU3.0 are either much faster or at worst equivalent. Especially, the improvement is more significant for large matrices such as ASIC_100ks and Raj1.

It is mentioned in Section 5.3.2 that stream mode starts when level size decreases to 16. This number is also selected based on experiment. The results can be found in Figure 5.12. For the purpose of making the figure more clear, instead of using all matrices used in previous experiments, only the ones that benefit significantly from stream mode are selected. In the figure, N stands for the threshold of level size where stream mode begins, and the values plotted are GPU kernel runtimes with different N compared with that with $N = 5$. It can be seen that the runtime keeps reducing until $N = 16$. Except matrix Raj1, experiments with all other matrices show slower or equivalent results for larger N, which proves that $N = 16$ is a good choice.

According to profiling results, there being unused warps is the main challenge for this problem. Actually the newly proposed three modes of kernels have greatly improved utilization of threads in SM, despite some remaining mismatch due to unpredictable sparsity pattern of the matrix. This hurts the performance of stream mode most significantly. As in other modes the warp occupancy is as high as 80%, while in stream mode the average is 40%. However, it is also worth noting that in the ending stage of factorization, as the submatrix size is decreasing, warp occupancy would drop naturally.

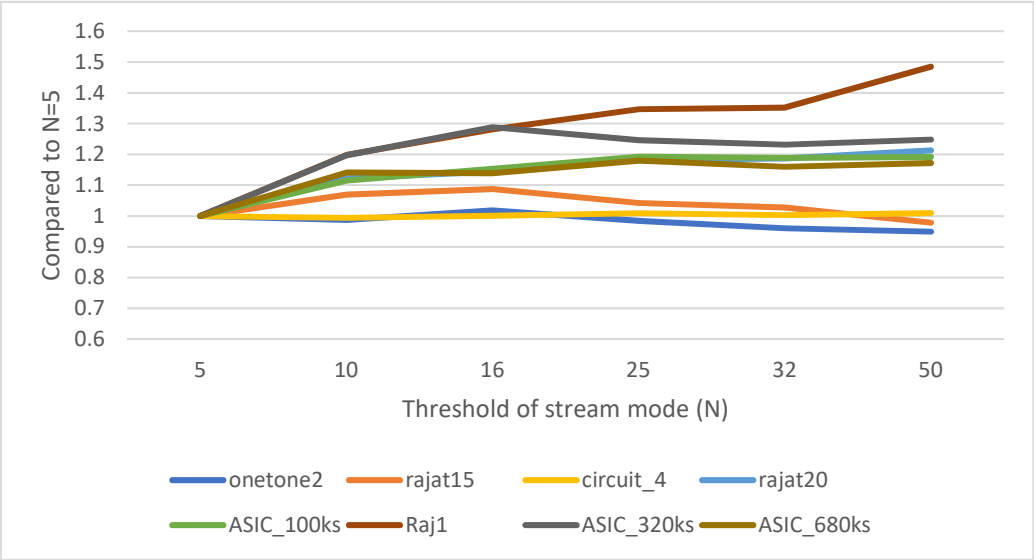


Figure 5.12: Performance of GPU kernel with different stream mode threshold settings

Note that driver overhead is also significant in many of the tests. Take ASIC_100ks as an example, the first cuda function call (including invisible set-up works) takes 40% of all GPU time (215ms). For larger matrices, this problem should be less severe for larger matrices or in real simulation scenarios where the factorization kernel is called repeatedly.

5.5 Summary

A new sparse LU solver on GPU for general scientific computing is developed. The new sparse LU solver, called GLU3.0 has two main improvements. First, a more efficient data dependency detection algorithm is introduced. Second, three different kernel operation modes are developed based on different number of columns in a level, as the LU factorization progresses. They enable dynamic allocation of GPU blocks to better balance the computing demands and resources during the LU factorization process. Numerical results on the set of typical circuit matrices from University of Florida Sparse Matrix Collection (UFL) have

shown that GLU3.0 can deliver 2-3 orders of magnitude speedup over GLU2.0 for the data dependency detection. Furthermore, GLU3.0 achieves 13.0 $\times$ (arithmetic mean) or 6.7 $\times$ (geometric mean) speedup over GLU2.0 and 7.1 $\times$ (arithmetic mean) or 4.8 $\times$ (geometric mean) over recent proposed enhanced GLU2.0 sparse LU solver on the same set of circuit matrices.

Chapter 6

Conclusions

Interconnect TDDB remains one of the important reliability issues for VLSI chips. Although there are a lot of existing works focusing on the physics of TDDB, the application of these physics models to practical TDDB evaluation in the scale of VLSI chip remains an open problem. This thesis introduces three related studies focusing on circuit level TDDB lifetime evaluation, and one study on GPU based LU factorization solver, which has broader applicability and is helpful to general TDDB simulation and evaluation flows. The contributions of each study is summarized in this chapter.

6.1 Fast TDDB analysis with EPG model

This work introduces a fast way to evaluate TDDB lifetime based on the EPG TDDB model. It is based on the observation that the minimum distance between wires is the most important factor of TDDB lifetime, and thus the simplified case of parallel wires can be used to evaluate other cases with the same minimum distance. By deriving the analytic

solution of the EPG model, several accelerated methods of calculating lifetime are proposed and tested. One method that best balances speed and accuracy is based on the observation that the location of the minimum concentration can be determined by the dominant terms and the TTF can be computed by using a few dominant terms. On top of this, the method is extended to cover time-varying stressing voltage, which is commonly seen in practical VLSI chips. The equivalent DC stressing voltage, parameterized by amplitude, duty cycle, and period for periodic stressing voltage waveforms, is calculated using regression based method. Numerical experiments validate the proposed analytic TDDB concentration and TTF formula and the equivalent DC stressing voltage compact model against the results from FEM analysis using COMSOL. It is further shown that the new compact TDDB model can lead to three orders of magnitude speedup with less than 1% error.

6.2 Full-Chip Wire-Oriented TDDB Analysis

This work proposes a interconnect TDDB analysis flow based on a newly-defined metric called TDDB Damage for each wire. This method takes into account complex geometries of the layout and the length effect of wires. Lifetime of wires and the chip can be easily derived from TDDB Damage. To calculate TDDB Damage, electric field needs to be evaluated, and FEM is used to get the strength of electric field across the layout. To mitigate the challenge that the complete layout is too large, the layout is partitioned into much smaller tiles and analyzed separately, which is the proposed method called layout partition. Since the approach is based on solving for electric fields, it can cover various electric field acceleration models commonly used and can also account for the non-uniformity of elec-

tric field in all layout patterns. The new method compares favorably in terms of accuracy against a recently proposed full chip TDDB method. This allows a better determination of risk and helps avoid overly pessimistic designs due to larger-than-required interconnect spacing.

6.3 Data-Driven Fast Electrostatics and TDDB Aging Analysis

This work proposes a method to use machine learning, specifically a CNN based neural network to solve TDDB analysis targeted electrostatics problems, and is thus very helpful to the TDDB lifetime analysis method proposed in the previous work. The proposed method first encodes the electrostatic problem to be solved into an image. The neural network mimics the much slower FEM solver. Taking the encode image as input, it outputs an image of electric potential distribution which is the inferred result. Training and testing are done on a dataset from a synthesized CPU chip. Once trained, the model is applicable to any synthesized layout of the same technology. Compared to the conventional FEM based solver, the proposed method achieves 138x speedup, while keeping 99% of the accuracy on potential analysis, and 97% for TDDB aging analysis.

6.4 GLU3.0: Fast GPU-based Parallel Sparse LU Factorization Solver

This work introduces GLU3.0, which is an update to GLU, a GPU based sparse LU solver for general scientific computing. GLU3.0 improves GLU mainly in two factors. First, a more efficient data dependency detection algorithm is introduced, which drastically improves the complexity of the preprocessing phase. Second, the GPU kernel that does the work of numerical factorization is updated. Three different kernel operation modes are developed based on different number of columns in a level, as the LU factorization progresses. They enable dynamic allocation of GPU blocks to better balance the computing demands and resources during the LU factorization process. Numerical results on the set of typical circuit matrices from University of Florida Sparse Matrix Collection (UFL) have shown that GLU3.0 can deliver 2-3 orders of magnitude speedup over GLU2.0 for the data dependency detection. Furthermore, GLU3.0 achieves 13.0 $\times$ (arithmetic mean) or 6.7 $\times$ (geometric mean) speedup over GLU2.0 and 7.1 $\times$ (arithmetic mean) or 4.8 $\times$ (geometric mean) over recent proposed enhanced GLU2.0 sparse LU solver on the same set of circuit matrices.

Bibliography

[1] Comsol multiphysics reference manual, version 5.3a, 2017. http://www.comsol.com.

[2] K.-H. Allers. Prediction of Dielectric Reliability From I-V Characteristics: Poole-Frenkel Conduction Mechanism Leading to $\sqrt{E}$ Model for Silicon Nitride MIM Capacitor. *Microelectronics Reliability*, 44(3):411–423, 2003.

[3] M. Bashir, Dae Hyun Kim, K. Athikulwongse, Sung Kyu Lim, and L. Milor. Backend Low-k TDDB Chip Reliability Simulator. In *IEEE Int. Reliability Physics Symposium (IRPS)*, pages 2C.2.1–2C.2.10, 2011.

[4] C. C. Chen and L. Milor. Microprocessor aging analysis and reliability modeling due to back-end wearout mechanisms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2065–2076, Oct 2015.

[5] F Chen, M Shinosky, B Li, J Gambino, S Mongeon, P Pokrinchak, J Aitken, D Badami, M Angyal, R Achanta, et al. Critical ultra low-k tddb reliability issues for advanced cmos technologies. In *Reliability Physics Symposium, 2009 IEEE International*, pages 464–475. IEEE, 2009.

[6] Fen Chen, O. Bravo, Kaushik Chanda, Paul S. McLaughlin, Timothy D. Sullivan, Jason Gill, James R. Lloyd, Rick Kontra, and John M. Aitken. A Comprehensive Study of Low-k SiCOH TDDB Phenomena and Its Reliability Lifetime Model Development. In *Proceedings of the 44th International Reliability Physics Symposium*, IRPS '06, pages 46–53, New York, NY, Mar. 2006. IEEE Press.

[7] Fen Chen, Carole Graas, Michael Shinosky, Chad Burke, Kai D Feng, Craig Bocash, and Ramachandran Muralidhar. A method for rapid screening of various low-k tddb models. In *Reliability Physics Symposium (IRPS), 2015 IEEE International*, pages 3A–1. IEEE, 2015.

[8] Fen Chen, Carole Graas, Michael Shinosky, Chuck Griffin, Roger Dufresne, Ronald Bolam, Cathryn Christiansen, Kai Zhao, Shreesh Narasimha, Chunyan Tian, et al. New breakdown data generation and analytics methodology to address beol and mol dielectric tddb process development and technology qualification challenges. In *Reliability Physics Symposium, 2014 IEEE International*, pages 3A–1. IEEE, 2014.

[9] I. C. Chen, S. Holland, and C. Hu. A Quantitative Physical Model for Time-Dependent Breakdown in SiO_2. In *IEEE Int. Reliability Physics Symposium (IRPS)*, pages 26–28, 1985.

[10] Xiaoming Chen, Ling Ren, Yu Wang, and Huazhong Yang. GPU-accelerated sparse LU factorization for circuit simulation with performance modeling. *IEEE Trans. on Parallel and Distributed Systems*. `http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2312199`.

[11] Xiaoming Chen, Yu Wang, and Huazhong Yang. An adaptive lu factorization algorithm for parallel circuit simulation. In *17th Asia and South Pacific Design Automation Conference*, pages 359–364. IEEE, 2012.

[12] Xiaoming Chen, Yu Wang, and Huazhong Yang. NICSLU: An adaptive sparse matrix solver for parallel circuit simulation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 32:261–274, February 2013.

[13] Xiaoming Chen, Wei Wu, Yu Wang, Hao Yu, and Huazhong Yang. An escheduler-based data dependence analysis and task scheduling for parallel circuit simulation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(10):702–706, 2011.

[14] D Yu Chenhan, Weichung Wang, and Dan'l Pierce. A cpu–gpu hybrid approach for the unsymmetric multifrontal method. *Parallel Computing*, 37(12):759–770, 2011.

[15] Tai-Yu Chou and Zoltan J Cendes. Capacitance calculation of ic packages using the finite element method and planes of symmetry. *IEEE transactions on computer-aided design of integrated circuits and systems*, 13(9):1159–1166, 1994.

[16] K. Croes, P. Roussel, Y. Barbarin, C. Wu, Y. Li, J. Bömmels, and Z. Tőkei. Low field TDDB of BEOL interconnects using ¿40 months of data. In *2013 IEEE International Reliability Physics Symposium (IRPS)*, pages 2F.4.1–2F.4.8, April 2013.

[17] K. Croes, C. Wu, D. Kocaay, Y. Li, Ph. Roussel, J. Bommels, and Zs. Tokel. Current Understanding of BEOL TDDB lifetime Models. *ECS Journal of Solid State and Technology*, 4:N3094–N3097, 2015.

[18] Kristof Croes, Ph Roussel, Yohan Barbarin, Chunlin Wu, Yunlong Li, Juergen Bömmels, and Zs Tőkei. Low field tddb of beol interconnects using ¿ 40 months of data. In *Reliability Physics Symposium (IRPS), 2013 IEEE International*, pages 2F–4. IEEE, 2013.

[19] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006.

[20] T. A. Davis and E. Palamadai Natarajan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Trans. Mathematical Software*, pages 36:1–36:17, September 2010.

[21] Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 2011.

[22] James W. Demmel, John R. Gilbert, and Xiaoye S. Li. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. *SIAM J. Matrix Analysis and Applications*, 20(4):915–952, 1999.

[23] ERIC EATON. Data structures and algorithms. 1983.

[24] Martin Gall, Kong Boon Yeap, and Ehrenfried Zschech. Advanced Concepts for TDDB Reliability in Conjunction with 3D Stress. In *Proc. AIP Conference*, pages 79–88, 2014.

[25] N Galoppo, N.K Govindaraju, M Henson, and D Manocha. LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware. In *2005 Proceedings of the ACM/IEEE Supercomputing (SC) Conference*, page 3, 2005.

[26] Thomas George, Vaibhav Saxena, Anshul Gupta, Amik Singh, and Anamitra R Choudhury. Multifrontal factorization of sparse spd matrices on gpus. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 372–383. IEEE, 2011.

[27] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput.*, pages 862–874, 1988.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. `http://www.deeplearningbook.org`.

[29] A. S. Grove. *Physics and Technology of Semiconductor Devices*. Wiley, 1967.

[30] K. He, S. X.-D. Tan, H. Wang, and G. Shi. GPU-accelerated parallel sparse LU factorization method for fast circuit analysis. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 24(3):1140–1150, March 2016.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Changsoo Hong, Linda Milor, and MZ Lin. Analysis of the Layout impact on Electronic Fields in Interconnects Structures using Finite Element Method. *Microelectronics Reliability*, 44:1867–1871, 2004.

[33] X. Huang, V. Sukharev, Z. Qi, T. Kim, and S. X.-D. Tan. Physics-based full-chip tddb assessment for beol interconnects. In *Proc. Design Automation Conf. (DAC)*, pages 1–6. IEEE, June 2016.

[34] International technology roadmap for semiconductors (ITRS) interconnect, 2015 edition, 2015. `http://public.itrs.net`.

[35] Dae-Hyun Kim, Shu-Han Hsu, and Linda Milor. Optimization of experimental designs for system-level accelerated life test in a memory system degraded by time-dependent dielectric breakdown. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(7):1640–1651, 2019.

[36] Taeyoung Kim, Xin Huang, Hai-Bao Chen, Valeriy Sukharev, and Sheldon X-D Tan. Learning-based dynamic reliability management for dark silicon processor considering em effects. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 463–468. EDA Consortium, 2016.

[37] Jeffrey C. K. Lam, Maggie Y. M. Huang, Tsu Hau Ng, Mohammed Khalid Bin Dawood, Fan Zhang, Anyan Du, Handong Sun, Zexiang Shen, and Zhihong Mai. Evidence of Ultra-Kow-k Dielectric Material Degradation and Nanostructure Alteration of the Cu/Ultra-Low-k Interconnects in Time-Dependent Dielectric Breakdown Failure. *Applied Physics Letters*, 102(2):022908, 2013.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, May 2015.

[39] Wai-Kong Lee, Ramachandra Achar, and Michel S Nakhla. Dynamic gpu parallel sparse lu factorization for fast circuit simulation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, (99):1–12, 2018.

[40] EG Liniger, SA Cohen, and G Bonilla. Low-field tddb reliability data to enable accurate lifetime predictions. In *Reliability Physics Symposium, 2014 IEEE International*, pages BD–4. IEEE, 2014.

[41] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. Eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. *ACM Trans. Des. Autom. Electron. Syst.*, 20(2), March 2015.

[42] Zhijian Lu, Wei Huang, John Lach, Mircea Stan, and Kevin Skadron. Interconnect lifetime prediction under dynamic stress for reliability-aware design. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 327–334. IEEE Computer Society, 2004.

[43] J. W. McPherson. Time Dependent Dielectric Breakdown Physics - Models Revisited. *Microelectronics Reliability*, 52(9):1753–1760, 2012.

[44] J.W. McPherson and H.C. Mogul. Underlying Physics of the Thermochemical E Model in Describing Low-Field Time-Dependent Dielectric Breakdown in SiO_2 Thin Films. *Journal of Applied Physics*, 84(3):1513–1523, 1998.

[45] Eugeniy Meshcheryakov. python-gdsii 0.2.1, 2011. https://pypi.python.org/pypi/python-gdsii/.

[46] R. Muralidhar, E. Wu, T. Shaw, A. Kim, B. Li, P. Mclaughlin, J. Stathis, and G. Bonilla. A stochastic model for impact of LER on BEOL TDDB. In *2017 IEEE International Reliability Physics Symposium (IRPS)*, pages DG–4.1–DG–4.4, April 2017.

[47] S.P. Murarka, M. Eizenberg, and A.K. Sinha. *Interlayer Dielectric for Semiconductor Technologies*. Elsevier, Boston, 2003.

[48] R. H. Myers and D. C. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley-Interscience, 2002.

[49] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[50] Shaoyi Peng, Ertugrul Demircan, Mehul D Shroff, and Sheldon X.-D. Tan. Full-chip wire-oriented back-end-of-line tddb hotspot detection and lifetime analysis. *Integration*, 2019.

[51] Shaoyi Peng, Wentian Jin, Liang Chen, and Sheldon X.-D. Tan. Data-driven fast electrostatics and tddb aging analysis. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, MLCAD '20, pages 71–76, New York, NY, USA, 2020. Association for Computing Machinery.

[52] Shaoyi Peng and Sheldon X-D Tan. Glu3. 0: fast gpu-based parallel sparse lu factorization for circuit simulation. *IEEE Design & Test*, 37(3):78–90, 2020.

[53] Shaoyi Peng, Han Zhou, Taeyoung Kim, Hai-Bao Chen, and Sheldon X-D Tan. Physics-based compact tddb models for low-k beol copper interconnects with time-varying voltage stressing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(2):239–248, 2018.

[54] Ling Ren, Xiaoming Chen, Yu Wang, Chenxi Zhang, and Huazhong Yang. Sparse LU factorization for parallel circuit simulation on GPU. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 1125–1130, June 2012.

[55] Horst Rinne. *The Weibull distribution: a handbook*. Chapman and Hall/CRC, 2008.

[56] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[57] O. Schenk, A. Waechter, and M. Hagemann. Matching-based Preprocessing Algorithms to the Solution of Saddle-Point Problems in Large-Scale Nonconvex Interior-Point Optimization. Journal of Computational Optimization and Applications. *Journal of Computational Optimization and Applications*, 36(2-3):321–341, 2007.

[58] B. I. Shklovskii and A. L. Efros. *Electronic Properties of Doped Semiconductors*. Springer-Verlag, 1984.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[60] N. Suzumura, S. Yamamoto, D. Kodama, K. Makabe, J. Komori, E. Murakami, S. Maegawa, and K Kubota. A new TDDB degradation model based on Cu ion drift in cu interconnect dielectrics. In *IEEE Int. Reliability Physics Symposium (IRPS)*, pages 26–30, 2006.

[61] T. L. Tan, C. L. Gan, A. Y. Du, and C. K. Cheng. Effect of Ta Migration from Sidewall Barrier on Leakage Current in Cu/SiOCH Low-k Dielectrics. *Journal of Applied Physics*, 106(4):043517, 2009.

[62] Wei Tang, Tao Shan, Xunwang Dang, Maokun Li, Fan Yang, Shenheng Xu, and Ji Wu. Study on a poisson's equation solver based on deep learning technique. In *2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, pages 1–3. IEEE, 2017.

[63] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3424–3433. JMLR. org, 2017.

[64] UMFPACK. http://www.cise.ufl.edu/research/sparse/umfpack/.

[65] M Vilmay, D Roy, C Monget, F Volpi, and JM Chaix. Copper line topology impact on the sioch low-k reliability in sub 45nm technology node. from the time-dependent dielectric breakdown to the product lifetime. In *2009 IEEE International Reliability Physics Symposium*, pages 606–612. IEEE, 2009.

[66] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[67] Terence KS Wong. Time dependent dielectric breakdown in copper low-k interconnects: Mechanisms and reliability models. *Materials*, 5(9):1602–1625, 2012.

[68] E Y Wu and J Sune. On Voltage Acceleration Models of Time to Breakdown—Part II: Experimental Results and Voltage Dependence of Weibull Slope in the FN Regime . *IEEE Transactions on Electron Devices*, 2009.

[69] Ernest Y Wu, WW Abadeer, Liang-Kai Han, Shin-Hsien Lo, and Gary R Hueckel. Challenges for accurate reliability projections in the ultra-thin oxide regime. In *Reliability Physics Symposium Proceedings, 1999. 37th Annual. 1999 IEEE International*, pages 57–65. IEEE, 1999.

[70] Kexin Yang, Taizhi Liu, Rui Zhang, and Linda Milor. A comprehensive time-dependent dielectric breakdown lifetime simulator for both traditional cmos and finfet technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, (99):1–13, 2018.

[71] K. B. Yeap, M. Gall, Z. Liao, C. Sander, U. Muehle, P. Justison, O. Aubel, M. Hauschildt, A. Beyer, N. Vogel, and E. Zschech. In Situ Study on Low-k Interconnect Time-Dependent-Dielectric-Breakdown Mechanisms. *Journal of Applied Physics*, 115(12):124101, 2014.

[72] Wenjian Yu and Xiren Wang. *Advanced field-solver techniques for RC extraction of integrated circuits*. Springer, 2014.

[73] Zhongyang Zhang, Ling Zhang, Ze Sun, Nicholas Erickson, Ryan From, and Jun Fan. Solving poisson's equation using deep learning in particle simulation of pn junction. In *2019 Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC)*, pages 305–308. IEEE, 2019.