UNIVERSITY OF CALIFORNIA
RIVERSIDE

Parallel CAD Algorithms and Hardware Security for VLSI Systems

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

Doctor of Philosophy

in

Electrical Engineering

by

Kai He

August 2016

Dissertation Committee:

Dr. Sheldon Tan, Chairperson
Dr. Qi Zhu
Dr. Daniel Wong

The Dissertation of Kai He is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

This thesis could not have been completed without the great support that I have received from so many people over the years. I wish to offer my most heartful thanks to the following people.

I would like to thank my advisor, Prof. Sheldon Tan for guiding and supporting me over years. He has set an example of excellence as researcher, mentor, instructor and role model. His kindness, insight and suggestions always lead me to the right way.

I would like to thank my thesis committee members, Prof. Qi Zhu and Prof. Daniel Wong for their direction, dedication and invaluable advice.

I would like to thank all the members in our VSCLAB. I thank especially Xuexin, Zao, Xin, Tan, Taeyoung, Haibao, Yan, David, Yue, Hengyang, Zhongdong, Chase and Zeyu for the collaborative research works, discussion and help, which lead to the presented works in this thesis. I appreciate the friendship of my fellow students in UCR.

Last but not least, I would like to thank my wife and my parents for the love, support, and constant encouragement during the years of my study. I undoubtedly could not have done this without them.

To my wife and my parents for all the support.

ABSTRACT OF THE DISSERTATION

Parallel CAD Algorithms and Hardware Security for VLSI Systems

by

Kai He

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, August 2016
Dr. Sheldon Tan, Chairperson

As integration scales to the 20nm regime and below, the integrated circuit (IC) design has seen the billion transistor counts. For instance, the latest Pascal GPU using 16nm FinFET technology from Nvidia has 150 billion transistors. As a result, it becomes very challenging to verify those billion-transistor chips and there is an urgent need to develop advanced and parallel simulation technique. On the other hand, the counterfeit ICs have become a major security threat for commercial and mission-critical systems. In addition to the huge economic impacts, they post significant security and safety threats on those systems. The objective of this thesis is to develop new techniques to address above two tough issues encountered in VLSI research: new fast parallel circuit simulation and potential solutions to mitigate the counterfeit IC problem.

To accelerate the circuit simulation, we study several important linear algebra operations in simulation steps, such as sparse matrix-vector multiplication (SpMV), direct linear LU factorization and iterative general minimum residual linear solver. Parallel computing such as general purpose GPU programs are good solutions for improving performance

of these operations. We apply GPU for these tasks and attain impressive speedup over traditional or CPU methods. All algorithms and implementations are demonstrated with representative numerical experiments and thorough comparisons among different methods and platforms.

For various types of counterfeit ICs - recycled, remarked, cloned, out-of-spec, and over-produced, we propose a multi-functional on-chip sensor and post-authentication policy for detecting and preventing them. Especially for recycled ICs, we propose two kinds of aging sensors, which are based on electromigration (EM)-induced aging effects and ring-oscillator (RO)-based frequency aging effects. These two aging sensors can effectively detect chip usage time for both short and long periods. Simulated results show the advantage of the proposed multi-purpose sensor against the existing on-chip sensors in terms of functionality, detection coverage and usage time estimation range and accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

1.1.1 Parallel Sparse Linear Algebra for Circuit Simulation

The verification of today's large linear global networks such as on-chip large power grid networks is very challenging for chip designers. Fast verification of voltage drops and other noises on power delivery networks is critical for final design closure. As the VLSI technology proceeds into sub-65 nm scale [4], one challenging job of power grid network is to predict and ensure a reliable on-chip power delivery.

Since the power grid network usually comes with a huge size, its simulation and verification take a lot of time, and sometimes even make the analysis completely failed. Intensive researches have been carried out to seek for efficient analysis of large power grid networks in the past decade. Various algorithms have been proposed to improve scalability in computing time and to reduce memory footprints [5, 6, 7, 8, 9]. But most of those tech-

niques are based on the homogeneous single-core architectures, which means their resource usage is still limited.

The course of computing has been permanently altered by the recent leap from single-core to multi-core or many-core technologies. Among them, the graphics processing unit (GPU), is one of the most powerful many-core computing systems arousing interests and input from both research and industry community [10]. Today, more and more high performance computing servers are equipped with GPUs as co-processors. These GPUs work in tandem with CPUs (on same computing node) connected by high-speed link like PCIe buses. GPU's massively parallel architecture allows high data throughput in terms of floating point operations (flops). For instance, the state-of-the-art NVIDIA Kepler K40c chip has a peak performance of over 4 Tflops performance in comparison with about 80–100 Gflops of Intel i7 series quad-core CPUs [11]. Currently, GPUs or GPU-clusters can easily deliver tera-scale computing, which was only available on super-computers in the past, for solving many large scientific and engineering problems.

Until now, dense linear algebra support on GPU is well developed, with its own BLAS library [12], but sparse linear algebra support is still limited. Modern NVIDIA GPUs are throughput-oriented many-core processors that can offer very high peak computational throughput. They favor computations exhibiting sufficient regularity of execution paths and memory access patterns. For sparse-matrix-based analysis, there are two kinds of solvers in general: the direct LU solver and the iterative solver. Although there are some recent efforts in this direction [13, 14], the sparse direct LU solvers on GPU is considered to be difficult due to the irregular structure of matrices and the complicated data dependency

during the numerical LU factorization. As a result, there remain a challenge for GPU-based fine-grained parallel LU solver. On the other hand, iterative solvers, which mainly depend on simple operations such as matrix-vector multiplication and inner product of vectors, are more amicable for parallelization, especially on GPU platforms. There are some newly published papers, such as [15, 16, 17, 18, 19], which confirm the practicality and effectiveness of iterative solvers in solving large linear dynamic networks like power grid networks.

Several research efforts have been proposed for parallelizing sparse LU factorization on shared memory multi-core CPU and GPUs. SuperLU [20, 21] implemented supernode-based Gilbert-Peierls (G/P) left-looking algorithm [22], and SuperLU_MT [21] is its multi-threaded parallel version developed for shared memory multi-core architectures. However, it is not easy to form super-node in some sparse matrix such as circuit matrix. KLU [23], which is specially optimized for circuit simulation, adopts Block Triangular Form based on G/P left-looking algorithm.

Recently, the KLU algorithm has been parallelized on multi-core architecture by exploiting the column-level parallelism [24, 25]. However, for parallel LU factorization solvers on GPU, existing works mainly focus on dense matrices including [26, 27, 28], very few works on the sparse matrix have been proposed. Ren *et al.* recently proposed a GPU-based sparse LU solver based on the G/P left-looking algorithm [14]. It exploits the column-level parallelism due to sparse nature of the matrix. The left-looking based method, which transforms the factorization computing into a number of triangular matrix solving, seems more efficient on GPU computing. But it possesses higher data dependency coming from solving the triangular matrices. The traditional right-looking LU factorization, which

involves only less data dependent vector operations, has not been well studied in GPU implementation.

There are also some research works for GPU-based iterative solver for sparse systems [29, 30, 31, 32, 33, 34, 35, 36]. In [30], GMRES solver has been accelerated on GPU by simply parallelizing the computing of polynomial preconditioners. In [31], Jacobi-preconditioned conjugate gradient algorithm is parallelized based block compressed row storage format. But this solver only works on single GPU and symmetric matrices. Work in [36] proposed a parallel GMRES based on existing GPU-enabled BLAS library [12]. Also a few existing works have been proposed to explore the hybrid accelerators such as GPUs and Xeon Phi.

1.1.2 Recycled Integrated Circuits – Detection and Avoidance

The counterfeiting and recycling of integrated circuits (ICs) have become major problems in recent years. These ICs potentially impact the security of electronic systems especially for military, aerospace, medical and other critical applications. In addition to diminishing system dependability and usability, counterfeiting reduces total revenue of companies from their research and development efforts, discourages innovation through the theft of intellectual properties (IPs), and produces low-quality products under established brand names [37]. According to couterfeit IC categorization and definition in [38], an electronic part is considered as as a counterfeit component which is not genuine if it is an unauthorized copy; or it does not conform to the original component manufacturer's (OCM) design, model, and/or performance; or it is not produced by the original component manufacturer or is produced by unauthorized contractors; or it is an off-specification, defective,

or used OCM product sold as "new" or working; or it has incorrect or false markings and/or documentation.

Today the most widely reported type of counterfeit parts is the recycled type. It is reported that in today's supply chain, more than 80% of the counterfeit components are recycled [39]. These used or defective ICs enter the market when electronic "recyclers" divert scrapped circuit boards away from their designated place of disposal for the purposes of removing and reselling the ICs on those boards. The recycling process involves removing ICs from the board or even dies in the ICs. There are several security issues associated with these ICs. Firstly, a used IC can act as a ticking time bomb [40] since it does not meet the specification of the OCM of the ICs; secondly additional die on top of the recovered die can carry a back-door attack, sabotage circuit functionality under certain conditions, or cause a denial of service [41].

The detection methods for recycled chips can be classified into physical methods and electrical methods [37]. Physical methods consist of incoming inspection methods such as visual inspection, X-ray imaging, package analysis method such as laser scanning microscopy, delid method, and the material analysis method such as using Fourier transform infrared, and X-ray fluorescence. Electrical methods contain the parameter tests, function tests, built-in tests and structural tests. In general, physical methods can be applied to all part types, but some of the methods are destructive and take hours to test. As a result, sampling is required to certify a batch of parts by observing a small number of parts. On the other hand, conventional electrical test methods are non-destructive and time efficient, yet they can be very expensive because such techniques are not necessarily designed for

counterfeit detection. Electrical test techniques are advantageous because the sampling is not required, and all parts can be tested. However, there are some issues associated with electrical tests that must be addressed.

In order to fast detect and effectively prevent the recycled chip, one viable approach is to insert a lightweight aging detecting sensor, which can directly tell the usage of the chips and some early efforts have been explored [42, 43, 44]. Method in [43] designed the ring-oscillator-based (OR-based) aging sensor that relies on the aging effects of MOSFETs to change a ring oscillator frequency in comparison with the reference one embedded in the chip. As the chip ages owing to the wear-out mechanisms such as negative biased temperature instability (NBTI) and hot carrier injection (HCI), the shift threshold voltage of MOSFET devices, thus the frequency of ring oscillator indicates the level of aging, and provides a simple readout of the value. However, this method can only give very rough estimation of the usage age of the chip as the shift of the frequency depends on many factors. In order to mitigate this problem, the antifuse-based (AF-based) sensor was developed in [37]. The AF-based sensor essentially is a counter, which counts the clocks or derivatives of the clock events to log the usage of the chip. The antifuse memory is used to make sure the data in the count will not be erased or altered by attackers. However, the AF-based sensors suffer large area overhead especially when more accurate usage is required [37]. Another problem with this method is that it may not reflect the true aging-dependent usage of a chip. For instance, it will log the same usage time for a chip for different on-chip temperatures, however, which can have dramatically impacts on the aging effects from electromigration, NBTI and HCI [45].

1.1.3 Goal and Dissertation Contributions

Implementation of efficient and scalable numerical methods on highly-parallel processors is an interdisciplinary task. Our aim is to provide new highly parallel schemes and algorithms for building different direct and iterative solvers for circuit simulation and more general scientific computing on CPU-GPU platforms. Special focus goes to a hybrid right-looking LU factorization algorithm and a general GPU-accelerated dynamic iterative solver with fast sparse matrix-vector multiplication.

On the other hand, for hardware security aspect, our goal is to design a lightweight on-chip sensor, which is based on electromigration-induced aging effects for fast detection and prevention of recycled ICs.

The contributions of this dissertation are summarized as follows:

- **GPU-accelerated sparse LU solver** We propose a new column-based right-looking LU factorization method, which is shown to be more amenable for exploiting the concurrency of LU factorization. The new method preserves the benefit of column-level concurrency and symbolic analysis in the left-looking method meanwhile, it allows more parallelism to be exploited. We show that the new *GLU* LU solver allows the parallelization of all three loops in the LU factorization on GPUs. In contrast, the existing GPU-based left-looking LU factorization approach can only allow two-level parallelization. We conduct comprehensive studies on the new GPU LU solver on a number of published general matrices, circuit matrices and self-made large RLC circuit matrices against some existing LU solvers to demonstrate the advantage of the proposed GLU solver.

- **Preconditioned GMRES iterative solver on CPU-GPU platform** We propose a new method called *GPU-GMRES*, which is based on the preconditioned Generalized Minimum RESidual (GMRES) iterative solver. We implemented it on heterogeneous CPU-GPU platforms with multiple GPUs. The proposed GPU-GMRES solver adopts a very general and robust incomplete LU based preconditioner with tunable fill-ins. We show that by properly selecting the right amount of fill-ins in the incomplete LU factors, a good trade-off between GPU efficiency and convergence rate can be made to achieve the best overall performance of the solver. Such tunable feature can make this algorithm very adaptive and flexible for different problems.

- **EM-based on-chip sensor for recycled IC detection** Instead of using traditional aging effects from devices (such as MOSFETs), the new EM-based aging sensor exploits the natural aging/failure mechanism of interconnect wires to time the aging of the chip. The new sensor is based on a newly proposed hydrostatic stress evolution model of EM effects for accurate prediction of the EM failure [46]. As a result, we can design the interconnect wire structures based on the copper interconnect technology so that the resulting wires can have detectable EM failure at a specific time with sufficient accuracy.

- **Multi-Functional on-chip sensor for comprehensive detection of couterfeit ICs** The proposed on-chip sensor can detect both recycled/remarked/out-of-spec chips, as well as cloned and over-produced ICs. The new on-chip sensor, which combines aging sensors with antifuse memory, can serve as a central on-chip security hardware IP for counterfeit IC detection, on-chip timer and post-fabrication authen-

tication and even activation module for ICs. On top of the new sensor hardware, we propose a post-fabrication authentication process to detect and prevent the non-detective counterfeit ICs.

1.1.4 Organization of the thesis

The rest of this thesis is organized as follows.

To carry out parallel linear sparse algebra for circuit simulation, Chapter 2 starts from CPU-GPU based programming architecture, which is used to accelerate our proposed solvers. Then, Chapter 2 describes a novel column-based right-looking LU factorization method, and hence results in an efficient direct solver for circuit simulation and general scientific computing. In Chapter 3 we present an efficient parallel dynamic linear solver, called *GPU-GMRES*, for transient analysis of large linear dynamic systems such as large power grid networks.

For the recycled ICs detection and avoidance, we present a new lightweight on-chip aging sensor in Chapter 4. The EM effect and physics-based EM model is reviewed. Then the aging sensor as well as the interconnect wire structures are presented. We investigate the parameters of the sensor and show the trade-off between accuracy and area cost. In Chapter 5, a multi-functional on-chip sensor for comprehensive detection of counterfeit ICs is discussed. We also propose a post-authentication policy for detecting and preventing all kinds of counterfeit ICs, including the recycled/ remarked/out-of-spec ICs, as well as cloned and over-produced ICs.

Finally, Chapter 6 concludes the thesis with brief summaries of the works.

Chapter 2

GPU-Accelerated Parallel Sparse LU Factorization

2.1 Introduction

Transforming a sparse matrix into its LU form is of crucial importance in linear algebra as it plays an important role in many numerical and scientific computing applications such as finite difference and finite element based methods. LU factorization operation represents the dominant computing cost in those problems and it is very important to improve the efficiency of the LU factorization algorithms. LU factorization for sparse matrices is the most important computing step for general circuit simulation problems for circuit designs. But parallelizing LU factorization on the popular many-core platforms such as Graphic Processing Units (GPU) turns out to be a difficult problem due to intrinsic data dependency and irregular memory access, which diminish GPU computing power.

Modern computer architecture has shifted towards the multi-core processor [47, 48] and many-core architectures [49]. The family of GPU is among the most powerful many-core computing systems in mass-market use [10]. For instance, the state-of-the-art NVIDIA Kepler K40 GPU with 2880 cores has a peak performance of over 4 TFLOPS versus about 80–100 GFLOPS of Intel i7 series Quad-core CPUs [50, 11]. In addition to the primary use of GPUs in accelerating graphics rendering operations, there has been considerable interest in exploiting GPUs for general purpose computation (GPGPU) [51].

Until now, dense linear algebra support on GPU is well developed, with its own BLAS library [12], but sparse linear algebra support is still limited. Modern NVIDIA GPUs are throughput-oriented many-core processors that can offer very high peak computational throughput. They favor computations exhibiting sufficient regularity of execution paths and memory access patterns. For sparse-matrix-based analysis, GPU acceleration has been applied to parallelize the shooting-Newton method for transient radio-frequency circuit analysis [52] and to speedup the generalized minimum residual analysis (GRMES) based iterative method for large-scale thermal analysis [36] in the past. However, parallelizing the sparse LU factorization operation is very difficult because of the irregular structure of matrices and the high data-dependency during the numeric LU factorization. As a result, they remain a challenge for GPU-based fine-grained parallel computing [50].

Several research efforts have been proposed for parallelizing sparse LU factorization on shared memory multi-core CPU and GPUs. SuperLU [20, 21] implemented supernode-based Gilbert-Peierls (G/P) left-looking algorithm [22], and SuperLU_MT [21] is its multi-threaded parallel version developed for shared memory multi-core architectures. However,

it is not easy to form super-node in some sparse matrix such as circuit matrix. KLU [23], which is specially optimized for circuit simulation, adopts Block Triangular Form based on G/P left-looking algorithm.

Recently, the KLU algorithm has been parallelized on multi-core architecture by exploiting the column-level parallelism [24, 25]. For parallel LU factorization solvers on GPU, existing works mainly focus on dense matrices including [26, 27, 28], very few works on the sparse matrix have been proposed. Ren *et al.* recently proposed a GPU-based sparse LU solver based on the G/P left-looking algorithm [14]. It exploits the column-level parallelism due to sparse nature of the matrix. The left-looking based method, which transforms the factorization computing into a number of triangular matrix solving, seems more efficient on GPU computing. But it possesses higher data dependency coming from solving the triangular matrices. The traditional right-looking LU factorization, which involves only less data dependent vector operations, has not been well studied in GPU implementation.

In this chapter, we propose a new sparse LU solver on GPUs for circuit simulation and more general scientific computing. The new method, called *GLU* method, is based on a hybrid right-looking LU factorization algorithm. We show that more concurrency can be exploited in the right-looking method than the left-looking method, especially on GPU platforms. We have the following contributions:

- We propose a new column-based right-looking LU factorization method, which is shown to be more amenable for exploiting the concurrency of LU factorization. The new method preserves the benefit of column-level concurrency and symbolic analysis in the left-looking method meanwhile, it allows more parallelism to be exploited.

- We show that the new *GLU* LU solver allows the parallelization of all three loops in the LU factorization on GPUs. In contrast, the existing GPU-based left-looking LU factorization approach can only allow two-level parallelization. We conduct comprehensive studies on the new GPU LU solver on a number of published general matrices, circuit matrices and self-made large RLC circuit matrices against some existing LU solvers to demonstrate the advantage of the proposed GLU solver.

Numerical results show that the proposed GLU solver can deliver 5.71× and 1.46× speedup over the single-threaded and the 16-threaded PARDISO solvers [53] respectively, 19.56× speedup over the KLU solver [23], 47.13× over the UMFPACK solver [54] and 1.47× speedup over a recently proposed GPU-based left-looking LU solver [14] on the set of typical circuit matrices from University of Florida Sparse Matrix Collection (UFL) [55]. Furthermore, we also compare the proposed GLU solver on a set of general matrices from UFL, GLU achieves 6.38× and 1.12× speedup over the single-threaded and the 16-threaded PARDISO solvers respectively, 39.39× speedup over the KLU solver, 24.04× over the UMFPACK solver and 2.35× speedup over the same GPU-based left-looking LU solver. Also comparison on self-generated RLC mesh networks shows a similar trend, which further validates the advantage of the proposed method over the existing sparse LU solvers.

This chapter is organized as follows. Section 2.2 reviews previous work that has been done to factorize sparse matrices into LU form on GPU, in particular the left-looking algorithm, GPU architecture and CUDA programming. In Section 2.3, we present the new column-based right-looking algorithm and its parallel implementation on GPU. Several numerical examples and discussions are presented in Section 2.4. Last, Section 2.5 concludes.

2.2 Review of LU factorization algorithms and CUDA

Before we present our new approach, we first review the two main-stream LU factorization methods: the left-looking G/P factorization algorithm [22] and a variant of the right-looking algorithms such as the Gaussian elimination method. We then review some recent works on LU factorizations on GPU and the NVIDIA CUDA programming system.

The LU factorization of a $n \times n$ matrix, A, has the form $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. For a full matrix, LU factorization has $O(n^3)$ complexity as it has three embedded loops.

2.2.1 Right-looking factorization method

The right-looking LU factorization is the traditional factorization including the Gaussian elimination method. The algorithm can be explained by the following equation:

$$\begin{bmatrix} l_{11} & \\ l_{21} & L_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ & U_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & A_{22} \end{bmatrix}, \tag{2.1}$$

where $l_{11} = 1$ is the a scalar, and l_{21} and u_{12} are the column and row vectors respectively, and L_{22} and U_{22} are the $(n-1) \times (n-1)$ submatrices. They can be computed by $u_{11} = a_{11}$, $u_{12} = a_{12}$, $l_{21} = a_{21}/u_{11}$. After this, we end up with a $(n-1) \times (n-1)$ equation to solve: $L_{22}U_{22} = A_{22} - l_{21}u_{12}$. The process repeats until we reach a 1×1 equation to solve. As we can see, the traditional right-looking method solves one row for U matrix and then one column for L matrix at each iteration. Then it updates the $(n-1) \times (n-1)$ submatrix A_{22} on the right part of the whole matrix and solves the reduced matrix recursively (so it is called the right-

looking method). Note that the right-looking method requires that A_{ii} is first factored before we can factor $A_{i-1,i-1}$, which indicates the sequential data dependency of this algorithm and its limits for potential parallel implementations (although the multifrontal based hierarchical schemes can be exploited for parallelization [56]). Note that we ignore all the reordering steps for fill-in reduction and numerical pivoting as well as symbolic analysis steps as we will visit them later.

2.2.2 Left-looking factorization method

The G/P left-looking method shows better performances for sparse matrices and easier implementation than the traditional Gaussian elimination based methods. It also allows the symbolic fill-in analysis of L and U matrices before the actual numerical computing. Instead of computing one row of U and one column of L, the left-looking method computes one column for both L and U instead. This is achieved by solving a lower triangular matrix. This lower triangular solution is repeated n times during the entire factorization (where n is the size of the matrix) and each solution step computes a column of the L and U factors. In this method, the matrix is traversed by columns from left to right. To compute current column, the algorithm has to look at all the previous computed columns on the left part of the L and U. So it is called left-looking method. Algorithm 1 shows one detailed implementation of the left-looking LU factorization. In this pseudo code, the current column is indexed by j, and the columns to the left of the current column are indexed by k. To compute the current column j, the algorithm looks left and finds all already factored column k ($k < j$), where $A_s(k, j) \neq 0$, and then uses these columns to update current column j. $A_s(x, y)$ indicates the LU symbolically factorized A matrix, where all the fill-ins

and non-zero elements are assigned with non-zero initial values and their memories are allocated. Fig. 2.1 illustrates the basic idea of the left-looking algorithm. The key operation of the left-looking algorithm is the triangular matrix solving, which is actually performed by the so-called vector multiple-and-add (MAD) operations sequentially.

One important observation for the left-looking algorithm is that since all the fill-in patterns of factored matrices are exploited, we know some columns can be solved independently and in parallel, which is called column-level parallelism in the existing approaches. Such concurrency does not exist in the existing traditional right-looking algorithms due to the recursive nature of the algorithm as we mentioned before.

Algorithm 1 The Gilbert-Peierls left-looking algorithm

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

2.2.3 Related works

The G/P left-looking method shown in Fig. 2.1 has been parallelized on GPU recently [14]. This method exploits the two-level concurrency in the left-looking algorithm due to the sparsity patterns of the matrices. First, it exploits the column-level parallelism in the left-looking algorithm as mentioned earlier. Based on the matrix sparsity pattern, the independent columns can be grouped into levels. So the outer j loop of Algorithm 1 can be parallelized by processing columns level by level. The so-called *cluster mode* in this

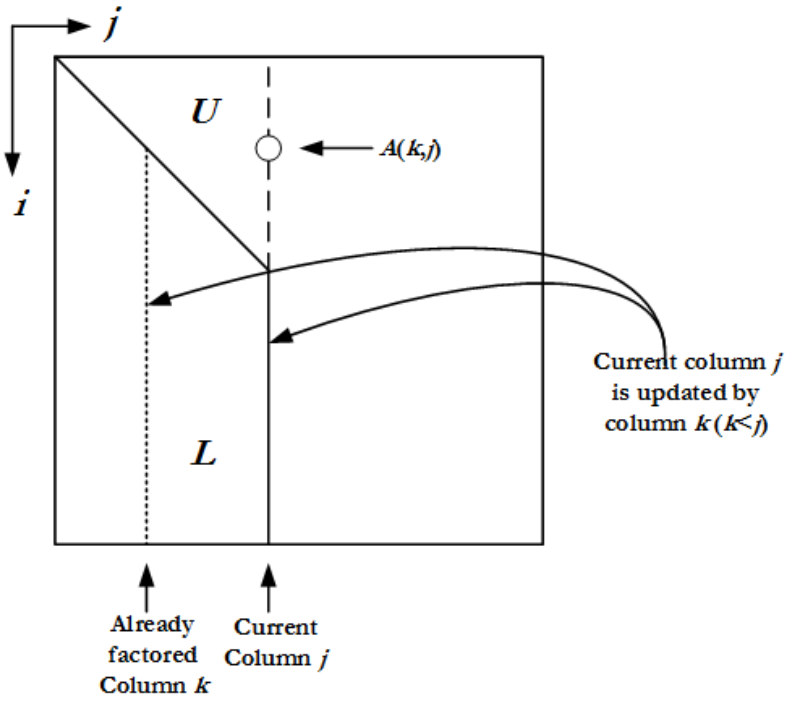


Figure 2.1: Left-looking update for column j

algorithm is for levels with many independent columns, while the *pipeline mode* is for levels with only a few columns. It also explores the parallelism within the vector MAD operation, which is reflected in the i loop of Algorithm 1. However, the middle k loop of column-by-column update, which is the key operation, is still in serial. The reason is that there is only one column j of the U matrix and updating this column must be done sequentially.

2.2.4 Review of GPU Architecture and CUDA programming

In this subsection, we review the 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 up to a huge amount of DRAM, referred to as global memory. Take the Tesla C2070 GPU for example. It contains 14 SMs, each of which has 32 streaming processors (SPs, or CUDA cores called by NVIDIA), 4 special function units (SFU), and its own shared memory/L1 cache. The structure of a streaming multiprocessor is shown in Fig. 2.2.

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. 2.3, 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

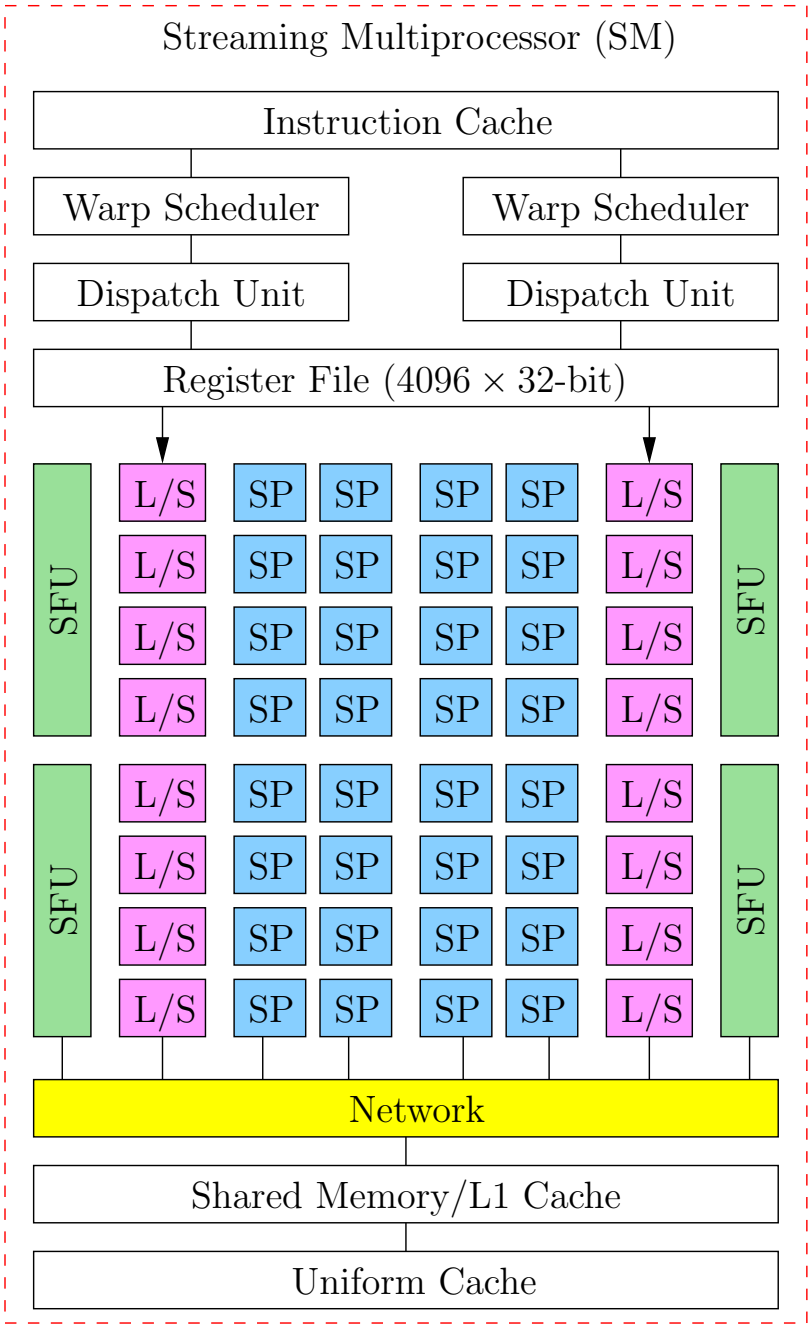


Figure 2.2: Diagram of a streaming multiprocessor in NVIDIA Tesla C2070. (SP is short for streaming processor, L/S for load/store unit, and SFU for Special Function Unit.)

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 responsible data elements. Fig. 2.3 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.

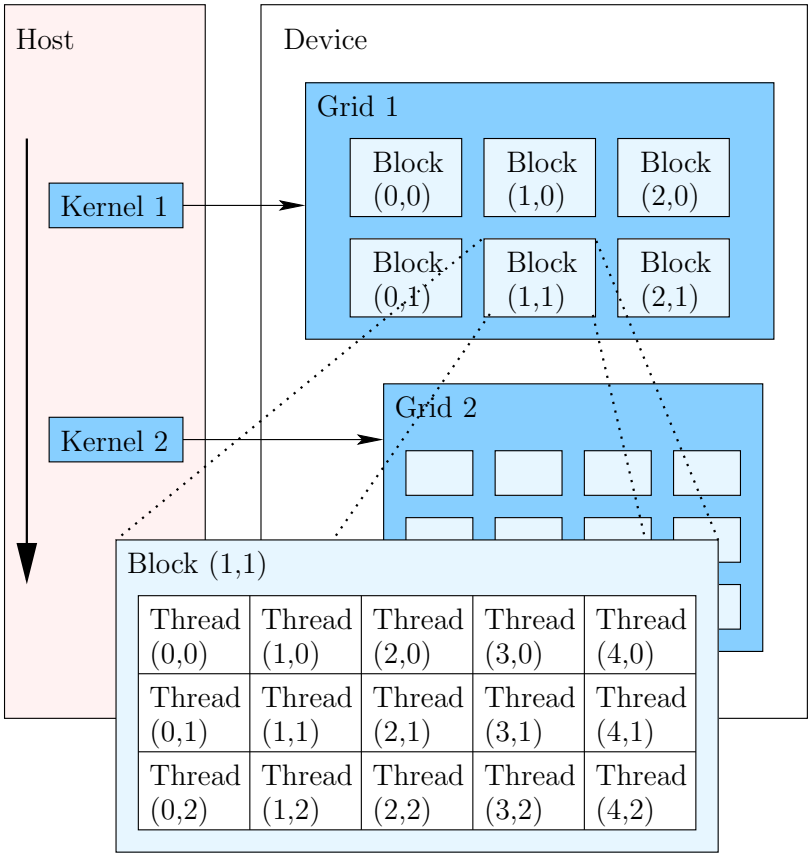


Figure 2.3: The programming model of CUDA.

2.3 Proposed GLU solver based on the hybrid column-based right-looking LU method on GPU platforms

In this section, we explain our new hybrid column-based right-looking sparse LU factorization method on the GPUs – GLU solver. GLU solver was originally inspired by the observation that the existing left-looking LU factorization has inherent limitations for concurrency exploitations due to the required solving of triangular matrices. To mitigate this problem, we look at the traditional right-looking LU factorization method, which seems more amenable for parallelization especially on GPU platforms. But we also want the benefits of symbolic analysis for storage management of factorized matrices and column-level concurrency in the left-looking based method. The resulting method is the hybrid column-based right-looking LU method, which will be discussed in the following.

2.3.1 The column-based right-looking algorithm

Our starting point is still the left-looking algorithm as we want to keep the column-concurrency and symbolic analysis and we still compute one column for both L and U matrices. But unlike the left-looking algorithm, once a column of L is computed, its impacts on the yet-to-be-solved columns will be updated right away (so we now start to look right in this sense). Algorithm 2 shows the hybrid column-based right-looking LU factorization algorithm, which turns out to be more amenable for GPU parallelization. In this pseudo code, the current column are indexed by k, and the columns to the right of the current column, which are updated immediately after the current column has been computed, are indexed by j. After current column k is computed, the algorithm looks right and finds all

column j ($j > k$) in the submatrix, where $A_s(k, j) \neq 0$, and then uses column k to update these columns. $A_s(x, y)$ indicates the LU symbolically factorized A matrix, where all the fill-ins and non-zero elements are assigned with non-zero initial values and their memories are allocated. Fig. 2.4 illustrates the basic idea of the hybrid column-based right-looking algorithm. The key operation of the right-looking algorithm becomes submatrix update now. But such change makes a major difference in terms of concurrency exploitation as we will show soon.

Algorithm 2 The hybrid column-based right-looking algorithm

1: **for** $k = 1$ to n **do**
2: /*Compute column k of L matrix*/
3: **for** $i = k + 1$ to n where $A_s(i, k) \neq 0$ **do**
4: $A_s(i, k) = A_s(i, k)/A_s(k, k)$
5: **end for**
6: /*Update the submatrix for next iteration*/
7: **for** $j = k + 1$ to n where $A_s(k, j) \neq 0$ **do**
8: **for** $i = k + 1$ to n where $A_s(i, k) \neq 0$ **do**
9: $A_s(i, j) = A_s(i, j) - A_s(i, k) * A_s(k, j)$
10: **end for**
11: **end for**
12: **end for**

In this column-based right-looking algorithm, we still have three loops. The outer k-loop chooses the current column k that will be factorized; the middle j-loop chooses the

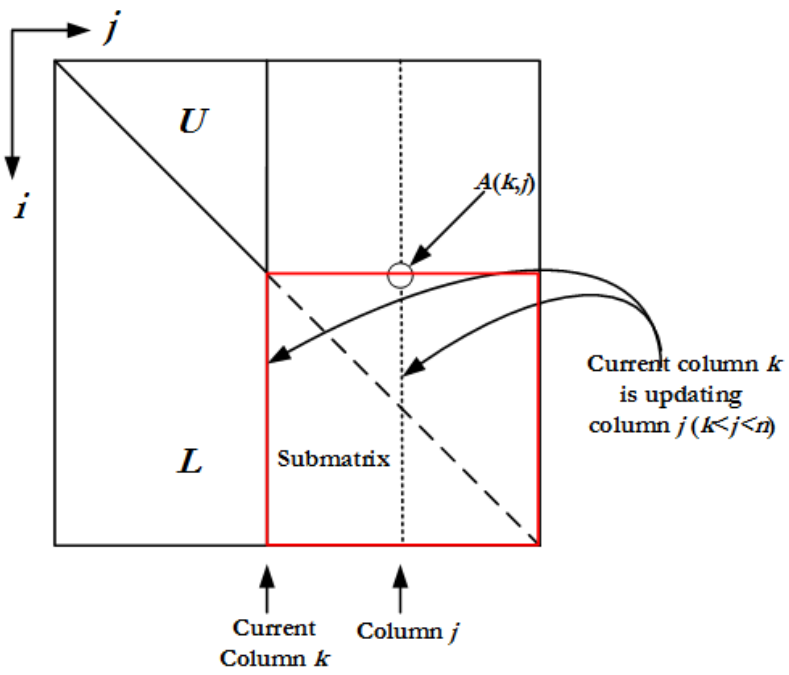


Figure 2.4: The illustration of the hybrid column-based right-looking algorithm and the submatrix update at each iteration

column j in the submatrix right to column k that depends on column k; and the inner i-loop is used to perform MAD operations between column k and column j. But now, we will show that these three loops can be parallelized because the submatrix is updated by the i and j loops, which is more amenable for parallelization than the solving triangular matrices in the left-looking method (see details in Subsection 2.3.3).

We remark the hybrid LU factorization method is similar to the multifrontal based right-looking LU factorization method, in the sense that each independent column and its connected columns can form a frontal matrix [56]. But in our approach, no elimination tree is used to build the frontal matrices and the hierarchical matrix analysis structure. The column-level parallelization is mainly based on the dependency graph (to be discussed later).

2.3.2 Preprocessing and symbolic analysis

As we mentioned earlier that the proposed method combines the benefits of both left-looking method and the right-looking methods. As a result, it still follows the preprocessing and symbolic analysis steps to improve the factorization efficiency. Hence the new factorization algorithm can still be split into three phases. In the sequel, we give a brief description of the first two steps for the self-contained purpose. Then we analyze the related data dependency from the symbolic analysis step for GPU computing.

First, the preprocessing phase preorders the matrix A to minimize fill-in and to ensure a zero-free diagonal. Second, the symbolic phase performs symbolic factorization and determines the structure of lower triangular matrix L and upper triangular matrix U. Then it groups independent columns into levels. Third, the numerical phase obtains the resulting lower and upper sparse triangular factors by solving the columns level by level. The preprocessing phase and symbolic phase are performed only once on CPU (which will be discussed in this section). The numerical phase can be performed multiple times on GPU. For the completion of the algorithm, we also present the first two phases.

In the preprocessing phase, we use HSL MC64 [57] to decrease the likelihood of encountering tiny pivots and AMD (Approximate Minimum Degree) algorithm [58] to reduce the fill-ins. The nonzero structure of the sparse matrix may dramatically change in course of LU factorization. In this step, we perform a left-looking algorithm based symbolic analysis [22] to determine the nonzero patterns of L and U. The core operation of left-looking algorithm is to solve the lower triangular system $L_k x = b$ in order to compute the kth column, where L_k is lower matrix representing the already computed $(k-1)$ columns

and the vector b is the kth column of matrix A. This pseudo core operation is shown in Algorithm 3.

Algorithm 3 Forward substitution for solving sparse triangular matrices

1: $x = b$

2: **for** $j = 0$ to $k - 1$ where $b(j) \neq 0$ **do**

3: **for** $i = j + 1$ to n where $L(i, j) \neq 0$ **do**

4: $x(i) = x(i) - L(i, j)x(j)$

5: **end for**

6: **end for**

From the pseudo code, we can see that entries in x can become nonzero in only two places, the first and fourth lines. We can represent these two relationships as a directed graph $G = (V, E)$, where the nodes $V = 1...n$ represent the rows and the edges $E = (j, i)$ where $L(i, j) \neq 0$. Thus, line 1 is equivalent to marking all nodes that are nonzeros in the vector b, whereas line 4 implies that if a node j is marked and it has an edge to a node i, then the latter must be also marked. Fig. 2.5 graphically highlights these two relationships. Therefore, the nonzero pattern can be computed by determining the nodes that are reachable from the nodes of vector b, which is also the computed column vector of U from the previous iteration of the left-looking method. This reachability problem can be solved using a classical depth-first search in G. Then we can determine the nonzero pattern of the new matrices L and U.

Fig. 2.6 shows a sparse matrix A and the predicted nonzero pattern of the LU factors of A after symbolic analysis (L and U share the same space of A), in which the

black circle and white circle represent original entries of A and fill-in entries respectively. As a result, the matrix A now contains the fill-ins and elements in the original matrix. When we copy A to GPU's memory, we actually copy the L and U matrices (with their values yet to be determined). After the factorization, the A becomes the resulting L and U matrices physically and it doesn't contain the original matrix any more.

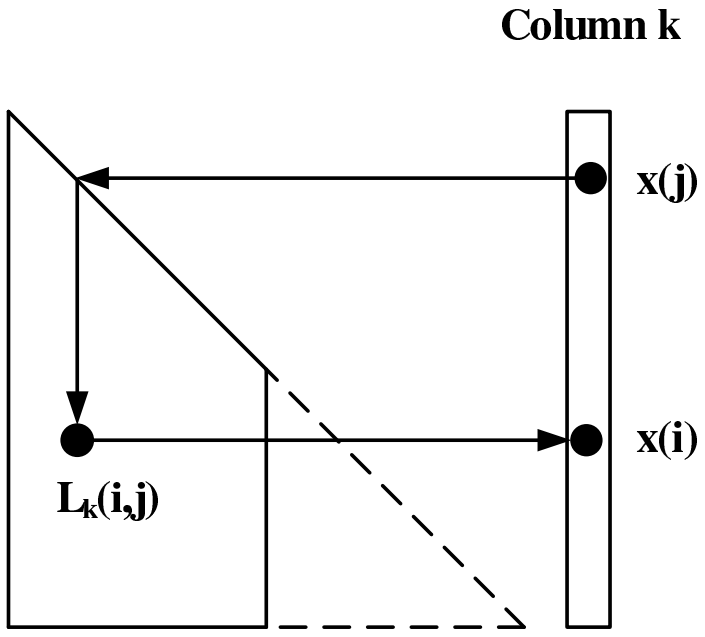


Figure 2.5: Nonzero pattern for a sparse triangular solver.

Another important problem is the column dependencies. It is clear that any column dependencies in the overall left-looking algorithm only arise from the *sparse triangular solve* step, the line 2 of Algorithm 3. However, when we compute column k, not all the columns to its left are needed, as it was illustrated in Algorithm 3. In fact, the factorization of column k only depends on the columns that satisfy $a_{ij} \neq 0$ for $i < j$. In other words, the dependencies between rows are defined by the sparsity pattern of the upper triangular

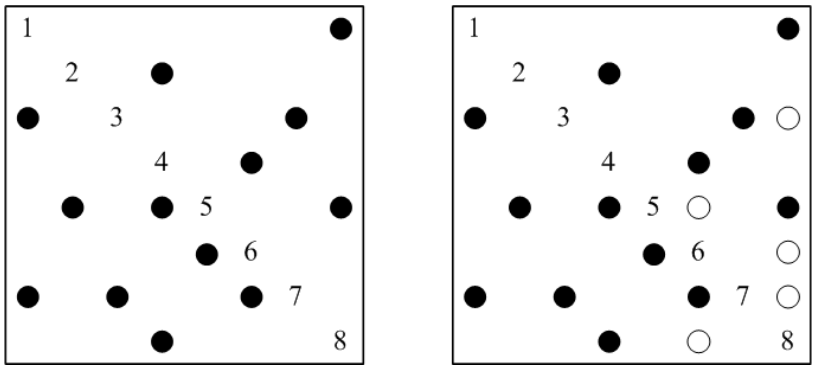


Figure 2.6: The original matrix A (left) and the matrix A (right) after symbolic analysis with predicted nonzero pattern of LU factors of A

matrix U and are independent of the lower triangular matrix L.

We use a directed acyclic graph (also called dependency graph) to represent the data dependencies in the LU factorization of the matrix in Fig. 2.6. In which, if column k depends on column i, then a directed edge exists from node i to node k, where $i < k$. Fig. 2.7 (top figure) illustrates the column dependencies of example matrix A. The graph was computed using predicted nonzero structure of matrix U only. All the columns in the same level are independent and can be computed in parallel. For instance, column 1, 2, 3, 5 can be evaluated in parallel; however, column 6 cannot be processed until columns 4 and 5 are computed.

Note that the concurrent computation resources (warps per streaming multiprocessor (SM), shared memory per block, threads per block) on GPU are limited. As a result, the number of columns, which can be solved in parallel, in each level should be limited. Hence, we propose a resource-aware levelization scheme, in which the number of columns of each level will be limited by a fixed number. For instance, Fig. 2.7 (bottom figure) shows the levelization result from the top figure in which the maximum number of allowed columns

is 3. This resource-aware levelization scheme will be applied to parallelize the outer k-loop of the proposed right-looking algorithm.

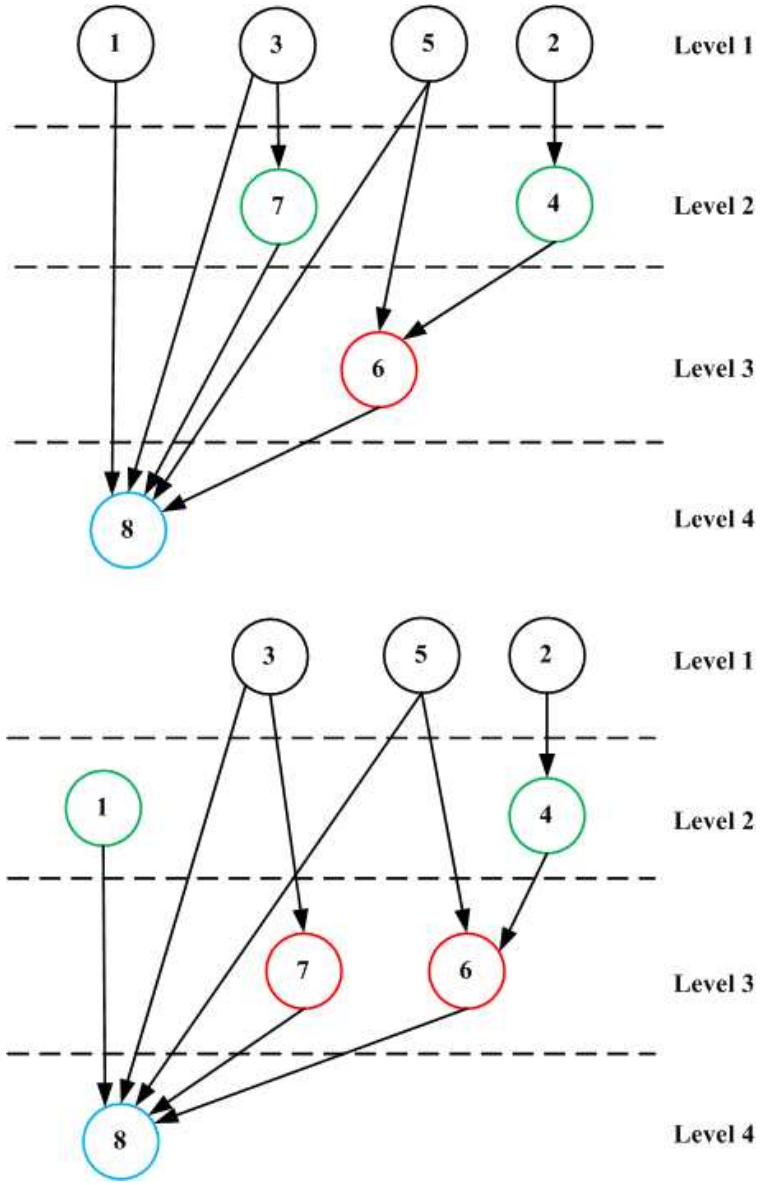


Figure 2.7: The illustration of the resource-aware levelization scheme.

2.3.3 Numerical computing phase

The algorithm based symbolic analysis can be altered to expose the column-level parallelism. Despite this exposed column-evaluation concurrency, in the numerical factorization phase, it is possible to explore more parallelism available in each level using multiple threads.

On GPU, the global memory access from the same CUDA warp (one warp consists of 32 threads and it is scheduling unit on one SM on GPU) can be coalesced if they are visiting the consecutive memory address. However, for sparse LU factorization, irregular nonzero pattern leads to many uncoalesced global memory accesses, which greatly degrades the performance. To maximize the coalescence, we use compressed sparse column (CSC) format to store the A matrices (L and U share the same storage of A) and record all nonzeros in L and U. In addition, to maximize parallelization during the factorization, we also use compressed sparse row (CSR) format to record the nonzero positions of symbolic U (but not its values), and its usage will become clear soon.

Now let us look at how the three loops in the proposed right-looking method can be parallelized in GPU platforms. Algorithm 4 is the pseudo code for the proposed parallel column-based right-looking algorithm. The first loop is to choose a number of columns of L matrix in one level, which can be factorized in parallel. Both the proposed method and the left-looking method enjoy this *column-level parallelism* as the proposed method is also based on the symbolic left-looking level analysis. The difference is in the other two loops of the two algorithms. Next, let us look at the computing steps inside the first loop (between line 2 and line 14). There are two stages. In the first stage, we compute the current column *col*

of the L matrix by vector-scalar division and it can be performed in parallel easily. Due to the first column-level parallelism, we may have several current column *col*'s to be updated.

In the second step, we perform the submatrix update (MAD operations) for the current column *col*. We are concerned *col*'s that are needed by other columns of the submatrix (means that $A(k, j) \neq 0$ in line 7 of Algorithm 2) . We call the columns in submatrix which depends on current column *col*, the *subcol*. To facilitate locating those *subcol*'s, we need to access the dependency graph, which is represented by the symbolic upper triangular matrix U. For instance, the *subcol*'s of a current column, say k, can be found by using the nonzero position information of row k of L. This also explains why we need to have symbolic U in the CSR format. Note that the current column *col* needs to be stored into a un-compressed array and the *subcol*'s can access the un-compressed array to get the *col* information to update themselves. Since the *subcol*'s only read information from the un-compressed column, there's no conflict.

Notice that each *subcol* only needs to be updated once by the current column. As a result, all *subcol*'s in one submatrix can be updated in parallel. This parallelism in the loop is called *submatrix update parallelism*. In the third loop, the core operation is vector MAD operation, which is used to update a *subcol*. In contrast, the current column *col* needs to be updated by all solved and relevant columns to its left in the left-looking algorithm in the left-looking algorithm and the updates to column k must be performed sequentially. Hence it cannot enjoy the *submatrix update parallelism*. As a result, we parallelize essentially all the loops in LU factorization in the proposed new method.

Algorithm 4 Parallel column-based right-looking algorithm on GPU

```
1: for level 1 to level m do
2:    /*column-level parallelism*/
3:    for all col's in current level in parallel do
4:       compute current col of L matrix
5:    end for
6:    synchronize threads
7:    for all col's in current level in parallel do
8:       /*submatrix update parallelism*/
9:       for all subcol's in current submatrix which depends on col in parallel do
10:         /*vector MAD operation parallelism*/
11:         update elements in one subcol
12:      end for
13:   end for
14:   synchronize threads
15: end for
```

2.3.4 Parallel implementation on GPU

In the parallel implementation of sparse LU factorization, the CPU is responsible for initializing the matrix and doing the symbolic analysis. The GPU only concentrates on the numerical factorization. The CPU is also responsible for allocating the device memory, copy the host inputs to the device memory, and copy the computed device results back to the host.

In the proposed algorithm, the matrix A is divided into several levels. With the resource-aware levelization scheme, the optimal number of thread blocks can be easily determined (see experimental section for some study results). We use one thread block to process one column in a level. In the first stage, we use one warp to compute one current column in the L matrix. Multiple current columns will lead to multiple blocks invoked in GPU in each kernel launch as each block can execute independently. Then we synchronize the threads within the block to ensure the current column is solved. In the second stage, we use one warp to update one *subcol*. There may be multiple active warps in one block now. Within one block, there is no data conflict among warps because they update different *subcol*. However, memory access conflict may occur between different blocks. For example, column j depends on both column k_1 and k_2, while column k_1 and k_2 are in the same level. In this case, the updates to the column j must be performed using atomic floating point operations. Finally, each element in *subcol*'s is updated by one thread. In this way, we can take the full advantage of the GPU powers. Fig. 2.8 illustrates the difference for concurrency exploitation and warp scheduling schemes between the left-looking and the proposed column-based right-looking algorithm. It can be seen that there is only one warp sequentially updating current column with the already factored columns in the left-looking algorithms. However, in the proposed right-looking algorithm, multiple warps can use the current column to update many different sub-columns concurrently.

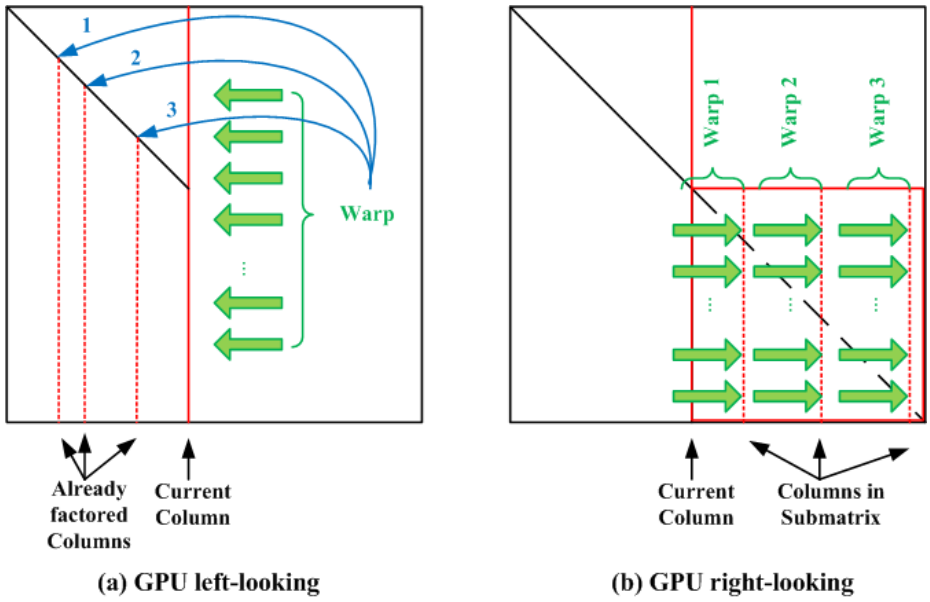


Figure 2.8: The comparison of the concurrency exploitation on GPU in terms of warp scheduling.

2.4 Numerical results and discussions

The proposed *GLU* LU factorization algorithm is implemented in C programming language. The GPU part is incorporated into the main program with CUDA C programming interface. The proposed method has been prototyped in CUDA 5.0 and the experimental results are carried out in a Linux server with two 8-Core Xeon E5-2670 CPUs, DDR3-1600 64GB memory. The server also consists of one K40 GPU and one K20 GPU, which serve as the GPU platforms for the proposed algorithms. Note that all the GPU results are obtained from the K40 GPU.

Table 2.1: The General Benchmark Matrices

Name	N	NNZ(A)	NNZ(A)/N	NNZ(L+U-I)
cage11	39082	59722	14.3	165367120
cant	62451	4007383	64.2	57830341
barrier2-11	115625	3897557	72.1	193180399
FEM_3D_thermal2	147900	3489300	23.6	93097092
thermomech_dK	204316	2846228	13.9	29085006
mc2depi	525825	2100225	4.0	54346411
epb3	643994	6175377	9.6	31698731
apache1	715176	4817870	3.9	8679783
ecology2	999999	4995991	5.0	45752523
thermal2	1228045	8580313	4.0	6330643

2.4.1 Performance comparisons

The benchmark matrices are listed in Table 2.1. The general matrices set and the circuit matrices set are from University of Florida Sparse Matrix Collection [55], which are used to evaluate the proposed GPU sparse LU factorization against other LU solvers. We also include a set of self-generated RLC mesh networks, which are used for providing comparison results on large circuit matrices. In this table, N is the matrix size, *NNZ(A)* means the number of nonzeros of the original matrix A, *NNZ(A)/N* shows the average number of nonzeros per row and *NNZ(L+U-I)* shows the number of nonzeros of the L and U matrices. Within each set, they are ranked with increasing number of N from top to bottom. Although our intention is for circuit matrices, we also include some matrices from wide applications to show that this GLU sparse solver can be applied for wide scientific and engineering applications.

We compare the proposed *GLU* solver against the recently proposed GPU left-looking algorithm (GPU-LL) [14], the UMFPACK solver [54], which is a right-looking multi-frontal solver, the KLU solver [23] and PARDISO [53], which is a state of the art parallel sparse LU solver, over the benchmark matrices in Table 2.1 and 2.3. In our performance evaluation, we use the CPU time reported by UMFPACK 5.6.2, KLU 1.2.0 and PARDISO 5.0.0.

Table 2.2 and 2.4 summarize the performance comparison results over benchmark matrices. And the listed time is only for numeric factorization, excluding preprocessing and symbolic analysis because the numeric factorization can be done many times in circuit

Table 2.2: The Performance Comparison Over General Benchmark Matrices

Bechmark name	GLU Runtime (s)	PARDISO T=1 (s)	PARDISO T=16 (s)	KLU Runtime (s)	UMFPACK Runtime (s)	GPU-LL Runtime (s)	Speedup over PARDISO T=1	Speedup over PARDISO T=16	Speedup over KLU	Speedup over UMFPACK	Speedup over GPU-LL
cage11	5.875	23.778	3.180	781.360	163.240	16.840	4.04	0.54	133.00	27.79	2.87
cant	3.662	3.543	0.491	33.741	18.1	11.03	0.97	0.13	9.21	4.94	3.01
barrier2-11	7.434	38.622	4.491	475.690	204.059	23.667	5.20	0.60	63.99	27.45	3.18
FEM_3D_thermal2	0.898	5.658	0.864	85.201	40.94	2.583	6.30	0.96	94.88	45.59	3.71
thermomech_dK	0.053	1.501	0.274	7.090	4.527	0.238	28.32	5.17	133.77	85.42	4.49
mc2depi	0.049	0.587	0.325	25.970	20.986	0.197	11.97	6.63	530.00	428.29	4.02
epb3	0.043	0.210	0.051	0.652	0.560	0.047	4.88	1.19	15.16	13.02	1.09
apache1	0.490	2.318	0.415	14.241	7.437	0.560	4.73	0.85	29.06	15.17	1.14
ecology2	6.940	6.604	0.928	38.131	25.330	8.990	0.95	0.13	5.49	3.64	1.30
thermal2	0.066	7.668	1.043	0.466	0.943	0.112	116.18	15.80	7.06	14.29	1.70
Arithmetic mean							18.36	3.20	102.16	66.56	2.65
Geometric mean							6.38	1.12	39.39	24.04	2.35

Table 2.3: The Circuit Benchmark Matrices

Name	N	NNZ(A)	NNZ(A)/N	NNZ(L+U-I)
Circuit matrices				
circuit_2	4510	21199	4.7	35612
rajat15	37261	443573	11.9	2028124
bcircuit	68902	375558	5.5	982513
ASIC_100ks	99190	578890	5.8	4271846
hcircuit	105676	513072	4.8	625958
scircuit	170998	958936	5.6	2518316
raj1	263743	1302464	4.9	10771367
ASIC_320ks	321671	1827807	5.7	4838888
rajat30	643994	6175377	9.6	31698731
ASIC_680ks	682712	2329176	7.2	4957172
G3_circuit	1585478	7660826	4.8	376618798
Freescale1	3428755	17052626	5.0	61281350
Self-generated RLC mesh networks				
rlc1	1970204	5930208	3.0	10169818
rlc2	3940404	11900408	3.0	21669968
rlc3	5890604	15731208	2.7	29761652
rlc4	15880404	47720408	3.0	81367568
rlc5	35621204	95082408	2.7	180026381

simulation and consume most of the simulation time while the other steps are not significant. For UMFPACK and KLU, it is difficult to see which one is better: KLU performs relatively better for circuit matrices, but are slower for matrices which take longer time to factor; UMFPACK beats KLU for general matrices which cost hundreds of seconds in their factorization, but is much slower for circuit matrices; single-threaded PARDISO performs very well on general matrices which are more dense than circuit matrices; The 16-threaded PARDISO is very fast on some of the cases but not so impressive on some large circuit matrices like *ASIC_680ks* and all of our self-generated RLC mesh networks; GPU-based left-looking solver, GPU-LL, has very stable performance on all of the three sets.

The proposed GLU algorithm outperforms the above solvers on most the matrices cases with various structures. For the general matrices set, compared to the KLU and UMFPACK, our speedup can achieve 39.4× and 24.0× on geometric mean, respectively. In addition, we can see in some cases such as *mc2depi*, the speedup over KLU solver can be as high as 530×. Compared to the single-threaded and the 16-threaded PARDISO solver, the speedup can be 6.38× and 1.12× on geometric mean, respectively. Compared to the GPU-LL solver, the speedup ranges from 1.09× to 4.49×, with 2.35× on geometric mean, which is still quite significant as the new solver is faster for all the matrices. On the other hand, we also notice that the speedup highly depends the structures of benchmark matrices.

Speedup in some cases such as *barrier2-11* is due to the fact that there are many denormal floating point numbers (extremely small real numbers) when factorizing this kind of matrix. CPU deals denormal numbers much slower than with normal represented numbers [59]. In contrast, the GPU can handle these numbers at the same speed as normal

Table 2.4: The Performance Comparison Over Circuit Benchmark Matrices

Bechmark name	GLU Runtime (s)	PARDISO T=1 (s)	PARDISO T=16 (s)	KLU Runtime (s)	UMFPACK Runtime (s)	GPU-LL Runtime (s)	Speedup over PARDISO T=1	Speedup over PARDISO T=16	Speedup over KLU	Speedup over UMFPACK	Speedup over GPU-LL
Circuit matrices											
circuit_2	0.008	0.002	0.004	0.004	0.006	0.011	0.25	0.50	0.54	0.78	1.43
rajat15	0.163	0.090	0.020	0.293	0.447	0.203	0.55	0.12	1.79	2.74	1.24
bcircuit	0.009	0.052	0.013	0.065	0.205	0.030	5.78	1.44	7.22	22.78	3.33
ASIC_100ks	0.031	0.292	0.090,	1.660	1.871	0.032	9.42	2.90	54.25	61.15	1.05
hcircuit	0.009	0.048	0.018	0.030	0.253	0.014	5.33	2.00	3.33	28.11	1.55
scircuit	0.056	0.13	0.031	0.339	0.829	0.106	2.32	0.55	6.05	14.80	1.89
raj1	0.189	0.355	0.078	73.842	125.799	0.211	1.88	0.41	390.69	665.60	1.12
ASIC_320ks	0.058	1.328	0.264	3.703	9.156	0.081	22.90	4.55	63.75	157.63	1.39
rajat30	1.234	6.309	1.468	0.317	230.59	1.864	1.19	0.26	15.58	186.86	1.51
ASIC_680ks	0.054	20.246	2.526	1.298	4.679	0.070	374.93	46.78	24.09	86.86	1.30
G3_circuit	0.672	26.464	3.467	516.882	133.358	1.054	39.38	5.16	769.16	198.44	1.57
Freescale1	0.235	4.004	0.689	13.486	67.414	0.284	17.04	2.93	57.45	287.20	1.21
Arithmetic mean							40.08	5.63	116.16	142.74	1.55
Geometric mean							5.71	1.46	19.56	47.13	1.47
Self-generated general RLC mesh networks											
rlc1	0.080	0.730	0.326	0.558	1.746	0.135	9.13	4.08	7.02	21.96	1.70
rlc2	0.180	1.567	0.419	1.263	6.499	0.267	8.71	2.33	7.04	36.21	1.49
rlc3	0.213	2.493	0.684	1.504	6.892	0.359	11.70	3.21	7.05	32.39	1.69
rlc4	0.626	5.770	1.545	4.638	**fail**	0.996	9.22	2.47	7.41	-	1.59
rlc5	1.274	15.539	3.279	9.807	**fail**	1.985	12.20	2.57	7.70	-	1.56
Arithmetic mean							10.19	2.93	7.24	30.19	1.61
Geometric mean							10.09	2.87	7.24	29.53	1.60

numbers [14]. So the GPU speedups for these matrices are very high. The performance comparison on these matrices clearly demonstrates the advantage of the proposed method.

We then compare the proposed method against other solvers on a set of typical circuit matrices which are also from UFL sparse matrix collection [55]. Compared with the single-threaded and the 16-threaded PARDISO, KLU, UMFPACK and GPU-LL, the proposed method achieves about 5.71×, 1.46×, 19.56×, 47.13× and 1.47× speedup, respectively, which further validates the advantage of the proposed method over the existing LU solvers. Please note that the proposed GLU can be slower than other solvers on very small circuit like *circuit_2* and *rajat15*. The possible reason is that the computation time is quite small and overheads become more significant, which was also observed in [60].

Last, we perform the comparison on a set of self-generated general RLC meshed networks. We notice that the KLU solver is highly optimized for the such circuit matrices and it indeed shows better performance. But still the 16-threaded PARDISO gives the best results among all the CPU sparse solver. However the new GLU method outperforms KLU about 7.24× on average. And the UMFPACK solver runs out of memory on two largest matrices. For the other 3 matrices, the proposed method also delivers about 29.53× speedup compared to UMFPACK solver. Also GLU outperforms single-threaded and 16-threaded PARDISO with 10.09× and 2.87× on average. Compared with the GPU-LL solver, GLU achieves about 1.60× speedup and it again consistently outperform the GPU-LL solver on all the examples, which further demonstrates the advantage of the proposed method over the existing GPU-LL method.

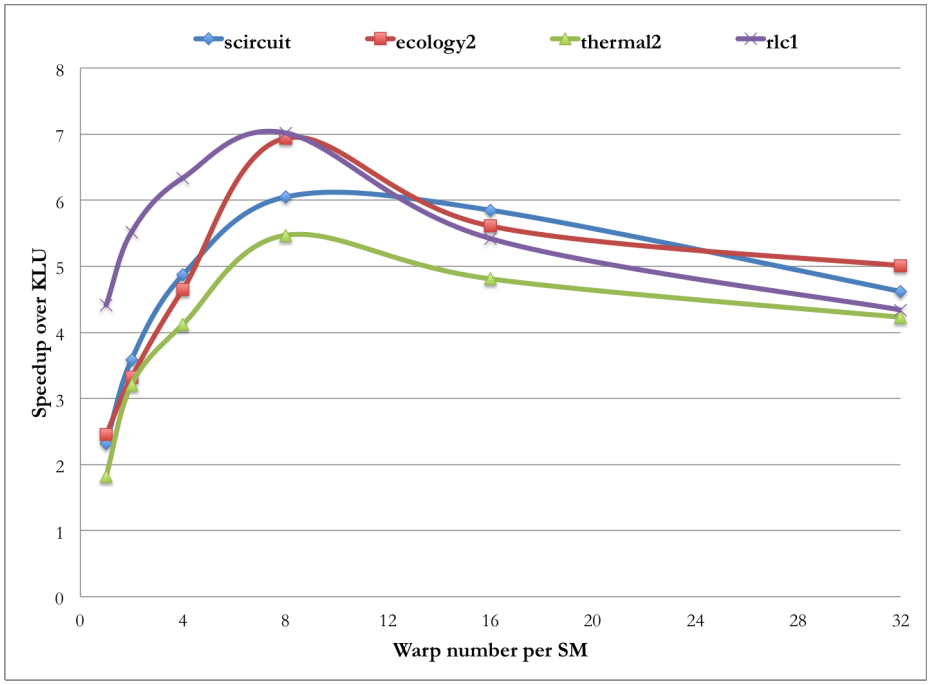


Figure 2.9: Speedup over KLU vs. number of warps per SM on K40c.

2.4.2 Impacts of warp number on performance

Next, we study one important design parameter and its impacts on performance of the GLU solver. We observe that one important parameter for the proposed solver is the number of warps allowed for each block or streaming multiprocessor (SM). Fig. 2.9 shows the speedup over KLU on four matrices on K40 GPU, with different number of warps per SM. The best performance is achieved when the number of resident warps per SM is around 8.

As we mentioned in Section 2.3.4, we use one warp to process one sub-column. Although more active warps can attain a higher parallelism, processing too many sub-columns simultaneously may decrease the performance. There are several reasons for this.

First, there may not be enough sub-columns as the matrix is sparse, while more resident warps mean more overhead on SM. Second, it may lead to more memory access conflicts as the threads need to perform slow atomic operations. With a threshold warp number set to 8, the performance reaches its best in terms of those trade-offs. When we further increase the number of warp, although there are more threads, the parallelism will not be further exploited so the performance starts to degrade. Note that such golden number of warp number depends on the structure of sparse matrix, and a few tries are needed to find the optimal number.

2.5 Summary

We have proposed a new sparse LU solver on GPUs for circuit simulation and more general scientific computing. The new algorithm is based on a hybrid right-looking LU factorization method, which we showed, is more suitable for GPU computing as it can exploit more parallelism than the widely used left-looking LU factorization algorithm. We further showed how the three loops of LU factorization can be parallelized based on the GPU thread structures, while the existing GPU left-looking LU factorization method can only parallelize two loops. Numerical results show that the proposed GLU solver can deliver 5.71× and 1.46× speedup over the single-threaded and the 16-threaded PARDISO solvers respectively, 19.56× speedup over the KLU solver, 47.13× over the UMFPACK solver and 1.47× speedup over a recently proposed GPU-based left-looking LU solver on the set of typical circuit matrices from University of Florida Sparse Matrix Collection (UFL). Furthermore, we also compare the proposed GLU solver on a set of general matrices from UFL, GLU achieves

6.38× and 1.12× speedup over the single-threaded and the 16-threaded PARDISO solvers respectively, 39.39× speedup over the KLU solver, 24.04× over the UMFPACK solver and 2.35× speedup over the same GPU-based left-looking LU solver. Also comparison on self-generated RLC mesh networks shows a similar trend, which further validates the advantage of the proposed method over the existing sparse LU solvers.

Chapter 3

Parallel GMRES Solver on GPU Platforms

3.1 Introduction

The verification of today's large linear global networks such as on-chip large power grid networks is very challenging for chip designers. Fast verification of voltage drops and other noises on power delivery networks is critical for final design closure. As the VLSI technology proceeds into sub-65 nm scale [4], one challenging job of power grid network is to predict and ensure a reliable on-chip power delivery. Since the power grid network usually comes with a huge size, its simulation and verification take a lot of time, and sometimes even make the analysis completely failed. Intensive researches have been carried out to seek for efficient analysis of large power grid networks in the past decade. Various algorithms have been proposed to improve scalability in computing time and to reduce

memory footprints [5, 6, 7, 8, 9]. But most of those techniques are based on the homogeneous single-core architectures.

The course of computing has been permanently altered by the recent leap from single-core to multi-core or many-core technologies. Among them, the graphics processing unit (GPU), is one of the most powerful many-core computing systems arousing interests and input from both research and industry community [10]. Today, more and more high performance computing servers are equipped with GPUs as co-processors. These GPUs work in tandem with CPUs (on same computing node) connected by high-speed link like PCIe buses. GPU's massively parallel architecture allows high data throughput in terms of floating point operations (flops). For instance, the state-of-the-art NVIDIA Kepler K40c chip has a peak performance of over 4 Tflops performance in comparison with about 80–100 Gflops of Intel i7 series quad-core CPUs [11]. Currently, GPUs or GPU-clusters can easily deliver tera-scale computing, which was only available on super-computers in the past, for solving many large scientific and engineering problems.

The NVIDIA CUBLAS libray [12] provides good dense linear algebra support on GPU, but the sparse linear algebra support is still limited. Although there are some recent efforts in this direction [13, 14], the sparse LU solvers on GPU is considered to be difficult due to the complicated data dependency. On the other hand, iterative solvers, which mainly depend on simple operations such as matrix-vector multiplication and inner product of vectors, are more amicable for parallelization, especially on GPU platforms. There are some newly published papers, such as [15, 16, 17, 18, 19], which confirm the practicality and effectiveness of iterative solvers in solving large linear dynamic networks

like power grid networks. But the existing approaches primarily focus on the multi-core CPU. The investigation of GPU power on these iterative solvers receives less attention.

Recently, there are also some research works for GPU-based iterative solver for sparse systems [29, 30, 31, 32, 33, 34, 35, 36]. In [30], GMRES solver has been accelerated on GPU by simply parallelizing the computing of polynomial preconditioners. In [31], Jacobi-preconditioned conjugate gradient algorithm is parallelized based block compressed row storage format. But this solver only works on single GPU and symmetric matrices. Work in [36] proposed a parallel GMRES based on existing GPU-enabled BLAS library [12]. Also a few existing works have been proposed to explore the hybrid accelerators such as GPUs and Xeon Phi.

In this chapter, an efficient parallel dynamic linear solver with application on transient analysis of large power grid networks of VLSI systems is proposed. We aim at developing a general GPU-accelerated dynamic linear solver, which not only can be applied to analyze power grid networks with different structures and properties, but also can be used to solve more general problems with asymmetric matrices. Examples include the power grid networks or thermal circuits with compact models, which may consist of controlled sources, constructed from model order reduction, subspace identification and other methods [61, 62]. Another example is the co-simulation of the power grids and voltage regulators. As a result, the new method, called *GPU-GMRES*, is based on the preconditioned Generalized Minimum RESidual (GMRES) iterative solver, and is implemented on heterogeneous CPU-GPU platforms with multiple GPUs. The proposed GPU-GMRES solver adopts a very general and robust incomplete LU based preconditioner with tunable fill-ins. We show that

by properly selecting the right amount of fill-ins in the incomplete LU factors, a good trade-off between GPU efficiency and convergence rate can be made to achieve the best overall performance of the solver. Such tunable feature can make this algorithm very adaptive and flexible for different problems.

In addition, since sparse matrix-vector (SpMV) multiplication is a key and the most time-consuming operation in a preconditioned GMRES solver, we also propose a new fast parallel SpMV algorithm on GPU platforms. The new algorithm, called *segSpMV*, reduces the memory access by partitioning the rows, whose nonzero patterns are irregular in general, into a number of fixed-length segments. As a result, the segSpMV method can enjoy the fully coalesced memory access and outperform existing GPU-enabled SpMV methods. To further improve the scalability and efficiency, *segSpMV* method is further extended to multi-GPU platforms, which leads to more scalable and faster *multi-GPU GMRES* solver. Furthermore, since many operations in the preconditioned GMRES solver such as SpMV and sparse triangular solving are bandwidth limited operations, it is important to reduce the data communication traffics. As a result, we properly partition the major computing tasks in the GMRES solver to minimize the data traffic between CPU and GPU, which further boosts performance of the proposed method.

Experimental results on the set of the published IBM benchmark circuits and mesh-structured power grid networks show that the *GPU-GMRES* solver can deliver order of magnitudes speedup over the director solver, UMFPACK [54]. The resulting *multi-GPU-GMRES* can also deliver 3-12× speedup over the CPU implementation of the same GMRES method on transient analysis. We also show that the matrix structures and property have

a huge impact on the efficiency of GMRES solvers. Note that some preliminary results of this article appeared in [63].

This chapter is organized as follows. Section 3.2 reviews power grid analysis problem and the GPU architecture. Section 3.3 describes the proposed GPU-GMRES parallel algorithm and discussions of ILU preconditioner. In Section 3.4, we present a new fast parallel SpMV algorithm and its implementation on multi-GPU platforms, followed by several numerical examples in section 3.5. Last, Section 3.6 concludes this chapter.

3.2 Review of power gird simulation and GPU architecture

3.2.1 The problem of power grid simulation

As shown in Fig. 3.1, a power grid network can be modeled as RLC (or RC) networks with known time-variant current sources, which can be obtained by gate-level logic simulations of the circuits. A typical power grid model has a tremendous size of over million nodes, and up to hundreds of thousands of input current sources. There are some nodes with known voltages in the grid, and are modeled as nodes connected with DC voltage sources. For C4 power grids, the known voltage nodes can be internal nodes inside the power grid.

The node voltages can be obtained by solving the differential equation which is formulated by modified nodal analysis (MNA),

$$\mathbf{G}\mathbf{x}(t) + \mathbf{C}\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{B}\mathbf{u}(t), \tag{3.1}$$

where $\mathbf{u}(t)$ is the given current source vector, $\mathbf{G} \in \mathbb{R}^{n\times n}$ is the conductance matrix,

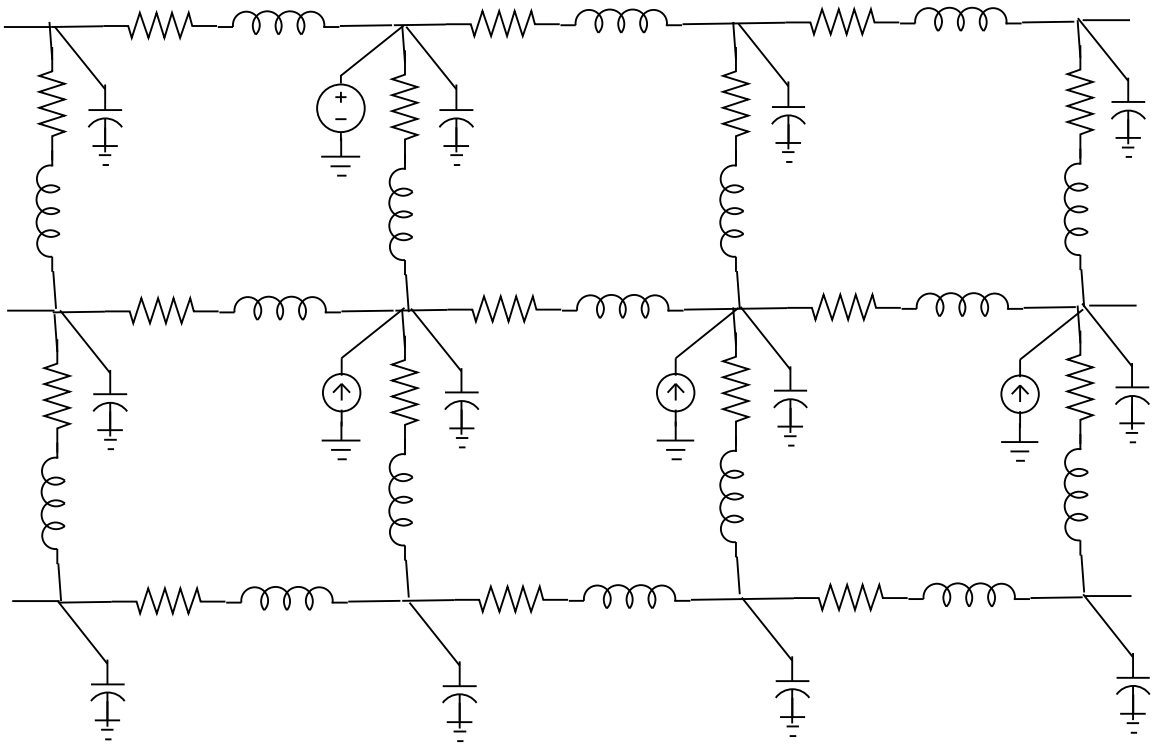


Figure 3.1: An RLC model of power grid network.

$\mathbf{C} \in \mathbb{R}^{n \times n}$ is the matrix resulting from charge storage elements, $\mathbf{B} \in \mathbb{R}^{n \times m}$ is the input selector matrix, $\mathbf{x}(t) \in \mathbb{R}^n$ is the vector of time-varying node voltages and branch currents of inductors and voltage sources, and $\mathbf{u}(t) \in \mathbb{R}^m$ is the vector of independent power sources. In general, the matrices $\mathbf{G}$ and $\mathbf{C}$ can be asymmetric. As a result, the conjugate gradient (CG) method may not be applied to the given problem. So we adopt the more general GMRES solver in this work to solve this problem.

With the backward Euler method, the transient behavior of the power grid can be solved step by step from a given initial condition $\mathbf{x}(0)$ using

$$(\mathbf{G} + \frac{1}{h}\mathbf{C})\mathbf{x}(t+h) = \frac{1}{h}\mathbf{C}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t+h), \tag{3.2}$$

where h is the time step length. If a fixed time step h is chosen, then the left-hand side matrix, $\mathbf{G} + \frac{1}{h}\mathbf{C}$, will remain the same along all time steps. Hence, when applying LU solver on this case, LU factorization only needs to be done once to obtain the LU factors of $\mathbf{G} + \frac{1}{h}\mathbf{C}$, and they can be reused for all the triangular solves in the following time steps.

3.3 Parallel GMRES solver on the GPU-CPU platform

3.3.1 ILU-based GMRES Solver

In general, our problem is how to solve a linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{3.3}$$

In our application, the coefficient matrix is $\mathbf{A} = \mathbf{G} + (1/h)\mathbf{C}$, and the right-hand side vector is $\mathbf{b} = (1/h)\mathbf{C} \cdot \mathbf{x}_{i-1} + \mathbf{u}_i$. The index of transient point is denoted by the subscript i , i.e., $\mathbf{x}_i = \mathbf{x}(t_i) = \mathbf{x}(i \cdot h)$. The linear equation like Eq. (3.3) can be solved by applying the LU factorization (direct) method or iterative methods. However, the implementations of LU-factorization solver are considered to be difficult on GPU due to many inherent data dependency and irregular memory access. On the other hand, the iterative solvers, are more amenable for GPU computing as only sparse matrix-vector (SpMV) and triangular matrix solving (in our implementation) operations are required, which are both GPU-friendly.

We investigate the GPU-accelerated GMRES iterative solver to solve the proposed power grid analysis problem. Considering the following system equivalent to $\mathbf{A}\mathbf{x} = \mathbf{b}$,

$$\mathbf{C}_{\mathrm{L}}\mathbf{A}\mathbf{C}_{\mathrm{R}}\mathbf{y} = \mathbf{C}_{\mathrm{L}}\mathbf{b}, \qquad \mathbf{y} = \mathbf{C}_{\mathrm{R}}^{-1}\mathbf{x}, \tag{3.4}$$

where $\mathbf{C}_{\mathrm{L}}$, $\mathbf{C}_{\mathrm{R}} \in \mathbb{R}^{n \times n}$ are non-singular. The $\mathbf{C}_{\mathrm{L}}$ and $\mathbf{C}_{\mathrm{R}}$ are referred as left preconditioner

and right preconditioner, respectively. The intuitive idea of preconditioning is to choose the matrices $\mathbf{C}_{\mathrm{L}}$ and $\mathbf{C}_{\mathrm{R}}$ so that $\mathbf{C}_{\mathrm{L}}\mathbf{A}\mathbf{C}_{\mathrm{R}}$ can approximate the identity matrix. This can be done by squeezing eigenvalues of $\mathbf{C}_{\mathrm{L}}\mathbf{A}\mathbf{C}_{\mathrm{R}}$ close to unity. In Eq. (3.4), we express this preconditioning process in the form of matrix multiplication. However, there can be other operations involved in practice. For instance, in our proposed solver, the two matrices $\mathbf{C}_{\mathrm{L}}$ and $\mathbf{C}_{\mathrm{R}}$ are actually the applications of lower and upper triangular solvers using the factors derived from incomplete LU factorization.

The combined efforts of the left factor and right factor in this splitting style preconditioning contribute to a more efficient GMRES, which is much better than using a single side precondition factor. Existing works have shown that such kind of simple preconditioners, e.g., diagonal (or Jacobi) preconditioner and approximate inverse preconditioner (AINV), do not have ideal preconditioning quality and they may even fail on some cases [64]. Moreover, very attractive preconditioners are defined in terms of an incomplete LU (ILU) factorization of $\mathbf{A}$. That is, we use the simplified version (or say, the cheaper variant) of LU method to compute $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, where $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ are sparse triangular matrices achieving the approximation $\mathbf{A} \approx \tilde{\mathbf{L}} \cdot \tilde{\mathbf{U}}$. Incomplete LU factorization is generally based on a modified Gaussian elimination, where the number of fill-in elements during factorization is strictly controlled below a preset limit. With row and column permutations, the generalized ILU can be used in most cases for preconditioning. With the application of permutation matrices, the preconditioned matrix system in Eq. (3.4) is

$$\mathbf{C}_{\mathrm{L}}\mathbf{A}\mathbf{C}_{\mathrm{R}} = (\tilde{\mathbf{L}}^{-1}\mathbf{P})\mathbf{A}(\mathbf{Q}\tilde{\mathbf{U}}^{-1}) \tag{3.5}$$

where $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ are ILU factors, and $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices. The construction

of the two ILU factors shall satisfy the approximation

$$\mathbf{PAQ} \approx \tilde{\mathbf{L}}\tilde{\mathbf{U}},$$

which is equivalently to say that $\tilde{\mathbf{L}}^{-1}\mathbf{PAQ}\tilde{\mathbf{U}}^{-1}$ is an approximation to identity matrix $\mathbf{I}$.

There is a critical trade-off between this approximation and the fill-in ratio of the ILU factors. A closer approximation needs more efforts in factorization and results in high fill-in ratio of LU factors. Therefore, it will incur a high computation cost during the preconditioning process, i.e., calculating Eq. (3.5). On the contrary, ILU factors with low fill-in ratio are cheap to be factorized, and they also require less effort in the triangular solves, but it could take more iterations in GMRES since the spectral property of the preconditioned system deteriorates. We will study this trade-off relationship in our experiment section.

The generalized minimum residual (GMRES) method is an iterative method for solving large-scale systems of linear equations ($\mathbf{Ax} = \mathbf{b}$), where $\mathbf{A}$ is sparse in our case. Algorithm 5 shows the standard Krylov-subspace based GMRES method with preconditioner [64], which uses projection method to form the m-th order Krylov-subspace [65, 64], e.g.,

$$\mathcal{K}_m = \text{span}(\mathbf{r}_0, \mathbf{MAr}_0, (\mathbf{MA})^2\mathbf{r}_0, \ldots, (\mathbf{MA})^{m-1}\mathbf{r}_0), \tag{3.6}$$

where $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ and $\mathbf{M}$ is the preconditioner. Note that for the sake of simplicity, we represent the ILU preconditioning process as an operation $\mathbf{M}$ here, and from now on, all the occurrence of $\mathbf{MA}$ should denote the operation in Eq. (3.5), which contains two sparse triangular solves and one sparse matrix vector multiplication. After orthogonalization and normalization, the orthonormal basis of this subspace is $\mathbf{V}_m$. To generate the Krylov subspace in GMRES, Arnoldi iteration is employed to form $\mathbf{V}_m$. Each Arnoldi iteration

generates a new basis vector and is appended to the previous Krylov subspace basis $\mathcal{K}_j$ to obtain the augmented subspace $\mathcal{K}_{j+1}$. Arnoldi iteration also creates an upper Hessenberg matrix $\tilde{\mathbf{H}}_m$ used to check the solution at the current iteration. As a result, the approximated solution $\mathbf{x}$ becomes the linear combination of $\mathbf{x}_m = \mathbf{x}_0 + \mathbf{V}_m \mathbf{y}_m$, where $\mathbf{y}_m$ is calculated in Line 12 of Algorithm 5.

The least squares problem is usually solved by computing the QR factorization of the Hessenberg matrix. In fact, the Hessenberg matrix can be maintained in factorized form by successively updating the factors. This procedure, which can be efficiently implemented by Givens rotations, is numerically reliable. However, the Gram-Schmidt orthogonalization inherent in Arnoldi method may be a source of numerical errors. Instead, we may use the modified Gram-Schmidt processes, or better, apply Householder transformations. The latter alternative is also well suited for parallel implementation.

3.3.2 Parallelization on GPU-CPU platforms

To parallelize the GMRES solver, we need to identify several computation intensive steps in Algorithm 5. There exist many GPU-friendly operations in GMRES, such as vector addition (`axpy`), 2-norm of vectors (`nrm2`), and SpMV multiplication (`segSpMV`). With preconditioning process, the triangular solves (`csrsv`) using ILU factors are also the beneficiaries of parallel computing, since many rows in ILU factors are independent and the solving of these rows can be done in parallel [66]. Based on the examples we focus on, we have noticed that SpMV multiplication and triangular solving take up to 70% of the overall runtime to build the Krylov subspace shown in Eq. (3.6). Those routines are GPU-

Algorithm 5 GMRES with left and right preconditioning.

Require: $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}_0 \in \mathbb{R}^n$ (initial guess), m (restart)

Ensure: $\mathbf{x} \in \mathbb{R}^n$: $\mathbf{A}\mathbf{x} \simeq \mathbf{b}$

1: $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$

2: $\tilde{\mathbf{r}}_0 = \mathbf{C}_{\mathrm{L}}\mathbf{r}_0$, $\beta = \|\tilde{\mathbf{r}}_0\|_2$, $\mathbf{v}_1 = \tilde{\mathbf{r}}_0/\beta$

3: **for** $j = 1, 2, \ldots, m$ **do** {Arnoldi iteration on GPU}

4: $\quad \mathbf{w} = \mathbf{C}_{\mathrm{L}}\mathbf{A}\mathbf{C}_{\mathrm{R}}\mathbf{v}_j$ {Eq. (3.5) using segSpMV and CUSPARSE csrsv}

5: $\quad$ **for** $i = 1, 2, \ldots, j$ **do** {using CUBLAS functions}

6: $\quad\quad h_{i,j} = \mathbf{w}_i^{\mathrm{T}}\mathbf{v}_j$

7: $\quad\quad \mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$

8: $\quad$ **end for**

9: $\quad h_{j+1,j} = \|\mathbf{w}\|_2$, $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$

10: **end for**

11: $\mathbf{V}_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$, $\tilde{\mathbf{H}}_m = \{h_{i,j}\}_{1 \le i \le j+1,\ 1 \le j \le m}$

12: $\mathbf{y}_m = \mathrm{argmin}_{\mathbf{y}} \|\beta\mathbf{e}_1 - \tilde{\mathbf{H}}_m\mathbf{y}\|_2$

13: $\mathbf{x}_m = \mathbf{x}_0 + \mathbf{C}_{\mathrm{R}}\mathbf{V}_m\mathbf{y}_m$

14: **if** not converge **then**

15: $\quad \mathbf{x}_0 = \mathbf{x}_m$, go to Line 1

16: **end if**

friendly (but they are bandwidth limited operations) and efforts have been made already to parallelize these routines in generic parallel algorithms for sparse matrix computations library CUSPARSE [67].

GPU programming for many engineering problems are typically limited by the data transfer bandwidth as GPU favors computationally intensive algorithms [50]. This is especially true for operations such as SpMV multiplication and sparse triangular solving, which are bandwidth limited. For instance, SpMV has $O(n)$ communication and $O(n)$ computing, so it has 1 to 1 computing and communication ratio (n is number of non-zero elements in the sparse matrices). Hence, it is important to reduce the data communication traffic for the proposed GPU-GMRES solver.

As a result, how to wisely partition the data between CPU memory (host side) and GPU memory (device side) to minimize data traffic is crucial for GPU computing. In the sequel, we make some detailed analysis first for GMRES in Algorithm 5. Although GMRES tends to converge quickly for most circuit examples, i.e., the iteration number $m \ll n$, the space needed to store the subspace $\mathbf{V}_m$ with a size of n-by-m, i.e., m column vectors with n-length, is still big. Therefore, transferring the memory of the subspace vectors between CPU memory and GPU memory is not an efficient choice. In addition, every newly generated matrix-vector product needs to be orthogonalized with respect to all its previous basis vectors in the Arnoldi processes. To utilize the data intensive capability of GPU, we keep all the vectors of $\mathbf{v}_m$ in GPU global memory. In this case, GPU is allowed to handle those operations, such as inner-product of basis vectors (`dot`) and vector subtraction (`axpy`), in parallel.

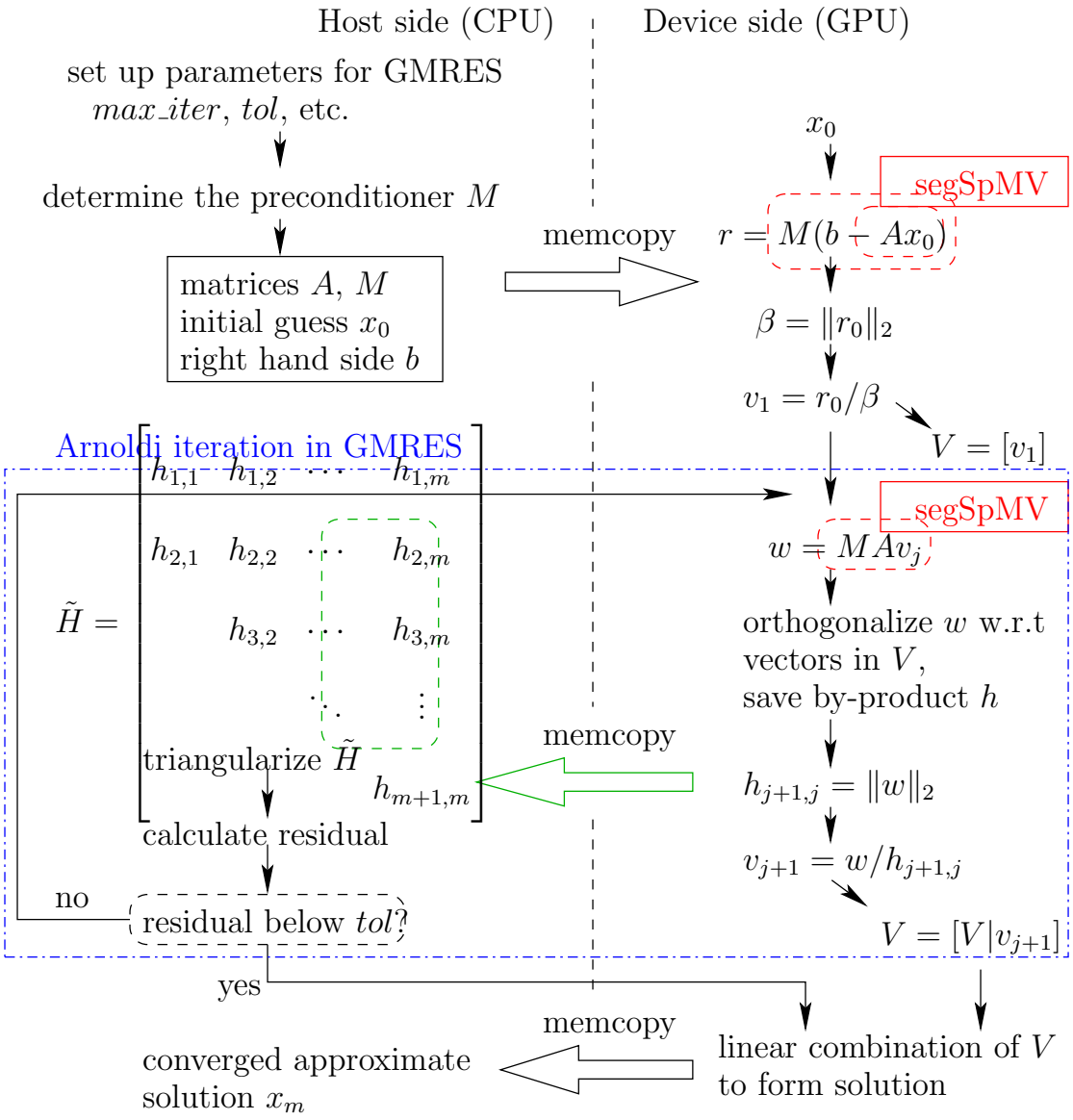


Figure 3.2: The proposed GPU-accelerated parallel preconditioned GMRES solver. We also show the partitioning of the major computing tasks between CPU and GPU here.

On the other hand, it is better to keep the Hessenberg matrix $\tilde{\mathbf{H}}$, where intermediate results of the orthogonalization are stored, at the CPU host side, because of the following reasons. First, its size is $(m+1)$-by-m at most, rather small if compared with circuit matrices and Krylov basis vectors. Besides, it is also necessary to triangularize $\tilde{\mathbf{H}}$ and check the residual in each iteration so the GMRES can return the approximate solution as soon as the residual is below a preset tolerance. Hence, in light of the sequential nature of the triangularization, the small size of Hessenberg matrix, and the frequent inspection of values by the host, it is preferable to allocate $\tilde{\mathbf{H}}$ in host memory. As shown in Algorithm 5, the memory copy from device to host is called each time when Arnoldi iteration generates a new vector and the orthogonalization produces a new vector $\mathbf{h}$, which is the $(j+1)$th column of $\tilde{\mathbf{H}}$, and is transferred to the CPU, where a least square minimization (a series of Givens rotations, in fact) is performed to see if the desired tolerance of residual has been met. Our observation shows that the data transfer and subsequent CPU based computation takes up less than 0.1% of the total run time.

Fig. 3.2 illustrates the computation flow, the partitions of the major computing steps and the memory accesses between CPU and GPU during the operations we mentioned above.

3.3.3 GPU-friendly implementation of preconditioners

One important aspect of the iterative solver is the preconditioner. Preconditioners increase the rate of convergence and thus reduce the number of iterations. A well chosen preconditioner will potentially make GMRES much faster than the one without preconditioner. In this section, we discuss the preconditioner for GPU GMRES.

We know from the preceding discussion that in ILU preconditioning process of Eq. (3.5), the two major participants are $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, who are sparse triangular matrices and approximate the $\mathbf{L}$ and $\mathbf{U}$ factors of $\mathbf{A}$ respectively. At the beginning of each Arnoldi iteration in Line 4 in Algorithm 5, this preconditioning procedure is needed to modify the property of a newly spanned Krylov subspace vector. For GMRES without preconditioner, Line 4 only consists a matrix-vector multiplication $\mathbf{A}\mathbf{v}_j$. In the new preconditioned GMRES solver, applying the ILU preconditioner requires two more operations: the solving of two sparse triangular systems (forward and backward substitutions).

For the two triangular ILU factors, we have two conflicting requirements. On one hand, the two triangular factors in ILU are supposed to approximate the complete LU factors as much as possible to increase the convergence rate. The more fill-in elements there are in $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, the more similarities there are between the preconditioned system in Eq. (3.5) and the identity matrix $\mathbf{I}$. Consider an extreme example in the other end. An ILU is called ILU0 if no fill-in elements are tolerated, and existing researches have shown that ILU0's applicability on many cases is very limited due to its poor performance in accelerating convergence of iterative solvers. On the other hand, when parallelizing the triangular solving of $\mathbf{L}$ and $\mathbf{U}$ matrices in GPUs, the efficiency of the GPU solver requires less data dependency (less dependency among rows) [66]. As a result, less fill-ins benefit GPU triangular solvers [68].

As a result, in this work, we adopt the strategy of ILU with fill-in ratio control. The ILU++ package we employ in our solver allows users to provide a threshold parameter and fill-in elements smaller than this threshold will be dropped off. This parameter gives us the

freedom to adjust and tune our ILU preconditioner, and delivers the optimal performance of the resulting GPU GMRES solver. But selection of the best threshold is still done by experiments and the best value is problem-specific in our work.

Once the circuit MNA matrix $\mathbf{A}$ is available, ILU is run to construct and set up the preconditioner. Then we transfer the matrices to GPU global memory. Before calling NVIDIA CUSPARSE's triangular solve function in calculating Eq. (3.5), there is one prerequisite step to analyze the structure of ILU factors $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$. According to CUSPARSE document, this step, which is called `csrsv_analysis`, makes an exploration of the matrix sparsity and the dependency between different rows (independent rows of triangular solve can be compute in parallel), so that information is collected and saved for future use in `csrsv_solve`. In a word, the analysis step is run only once for the whole simulation. The triangular solves in all GMRES iterations and all transient steps of circuit simulation can reuse this analysis information, and each time only `csrsv_solve` is called. More details will also be described in experimental section.

3.4 Parallel SpMV algorithm on the GPU-CPU platform

As we can see, in the preconditioned GRMES solver, one key computing step is the SpMV multiplication. In this section, we present the new GPU-accelerated parallel SpMV multiplication method, *segSpmV*.

3.4.1 Review of existing GPU-enabled SpMV algorithms

There are many sparse matrices formats such as DIA, ELL, CSR, HYB, PKT, COO with applications ranging from highly structured matrices (DIA, ELL) to unstructured matrices (HYB, COO) [69]. Among them, the compressed sparse row (CSR) can be used for both structured and unstructured sparse matrices and has wide application for sparse matrix computations. The CSR format is a popular, general-purpose sparse matrix representation. CSR explicitly stores column indices and nonzero values in arrays *col_idx* and *data*. A third array of row pointers, *row_ptr*, takes the CSR representation as shown in Fig. 3.3 for a 5×5 sparse matrix. For an $M \times M$ matrix, the *row_ptr* with length $M + 1$, stores the offset into *data* for the start point of each row, with the convention that $row\_ptr[M] = N_{nz}$, where N_{nz} is the number of nonzeros in the matrix.

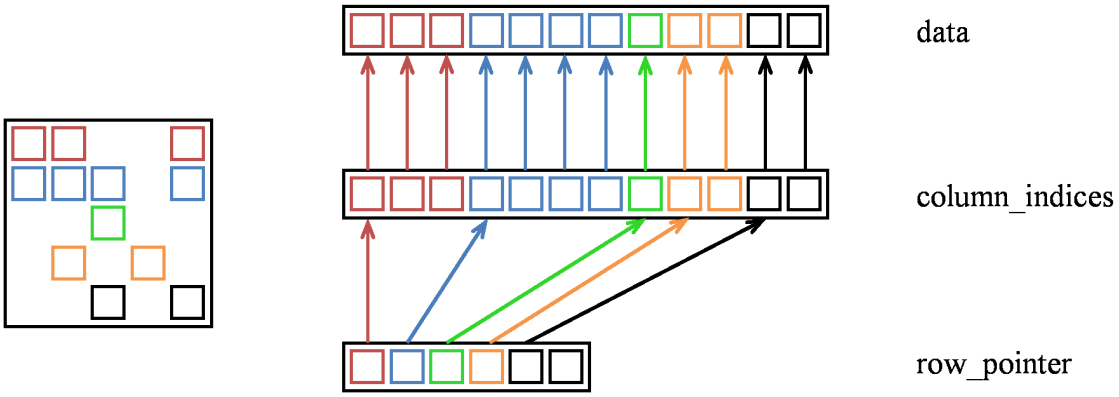


Figure 3.3: The CSR format of a sparse matrix

The SpMV computation consists of two phases: the first *product* phase, which performs the element-element production between the matrix and the vector, the second

summation phase adds the results for each row to get the final result. Several relevant SpMV algorithms on GPU platforms can be summarized as follows:

The row-based *B&G* method

Bell and Garland [69] first propose a straightforward implementation, in which each row will take care of all the computing (multiplication and summation) by a single thread as shown in Fig. 3.4. The algorithm only requires one kernel launch (one kernel launch means one CPU-to-GPU invocation). The main drawback of this approach is that each thread will read many sequential data from a *data* vector in the CSR format from the global memory of GPUs, which leads to slow non-coalesced memory access.

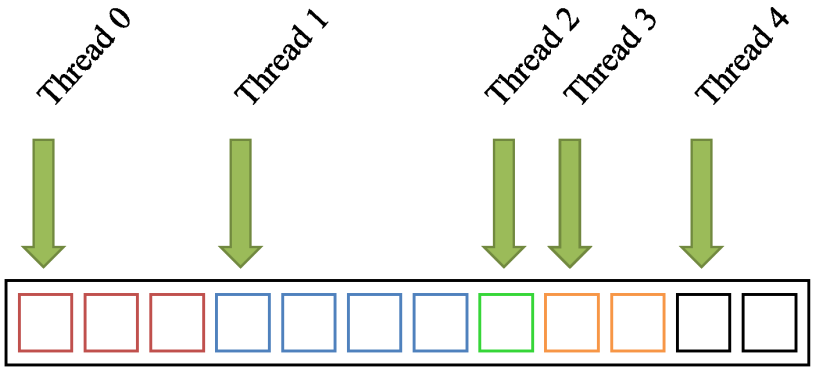


Figure 3.4: The illustration of the row-based *B&G* algorithm

The warp-based *B&G* method

The row-based *B&G* method is further improved by the warp-based *B&G* method [69] in which one warp is assigned to each row of a matrix. After the multiplication phase, the warp reduction is performed to compute the summation result. The algorithm is illustrated

in Fig. 3.5. Compared to the row-based $B\&G$ method, its memory accesses can be coalesced because 32 continuous threads in the same warp could work together to load the non-zero elements in one row. This method, however, may suffer from low performance when the number of nonzeros in each row is smaller than 32, which can be the case for many finite difference and finite element based methods.

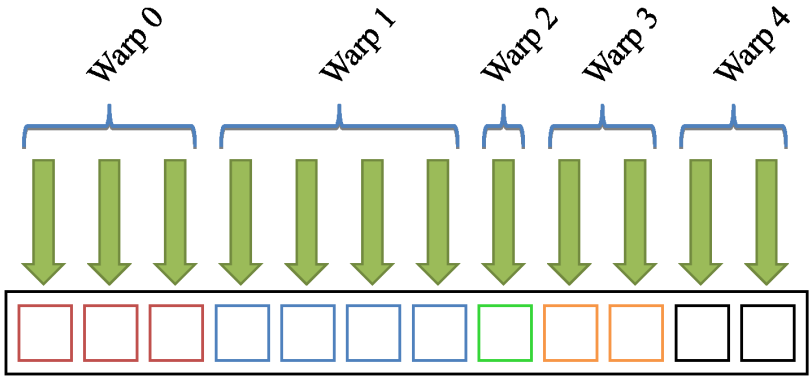


Figure 3.5: The illustration of the warp-based $B\&G$ algorithm

The $P\&S$ method

Deng *et al* later proposed an improved SpMV method, called $P\&S$ method, for many electronic design automation (EDA) related problems [35] . The approach will not directly operate on the CSR data structure. Instead, it creates a new vector, called *expanded vector*, of the same size of the *data* first as shown in Fig. 3.6. The *expanded vector* consists of the elements from the multiplication vector $[b1, ..., b3]$. And each element in the vector, *expended_vector[i]*, corresponds to one element in *data*, which is *data[i]*, and both of them will be multiplied in the production phase.

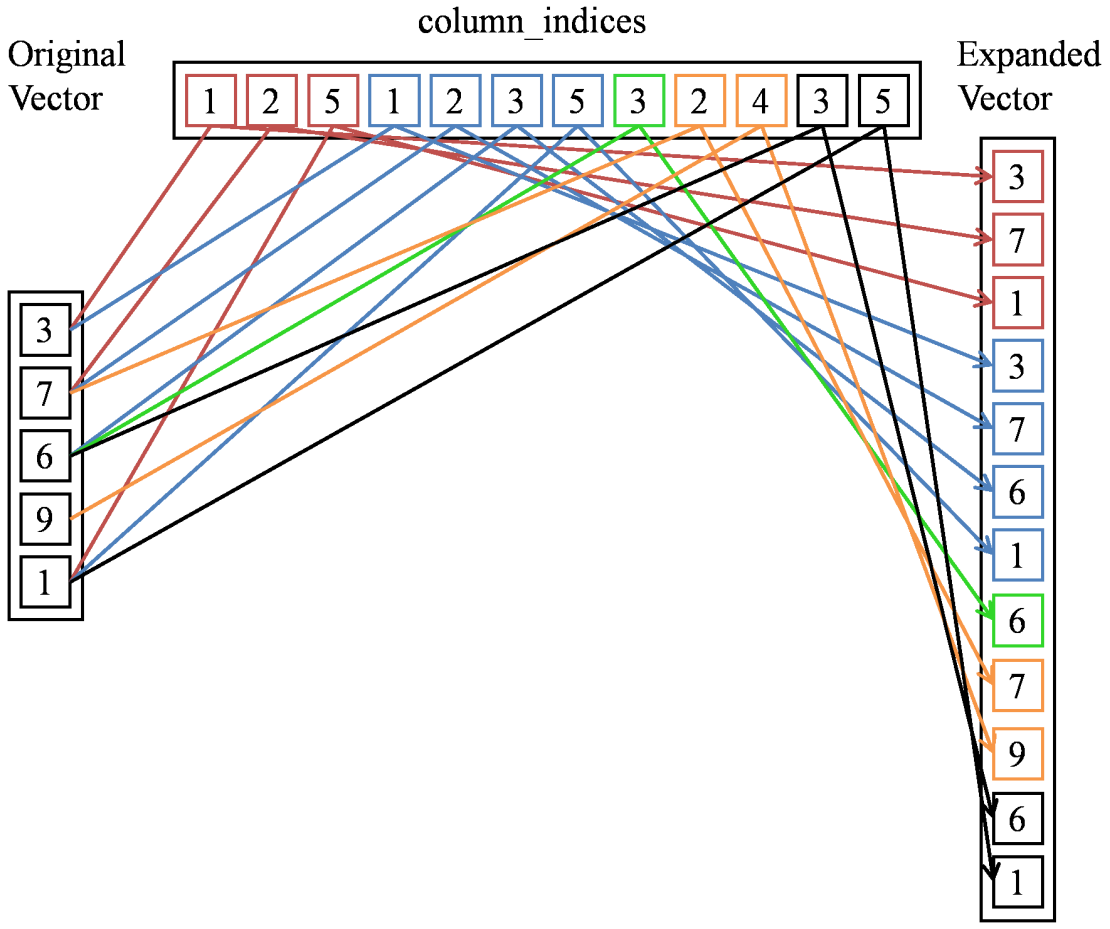


Figure 3.6: The vector expansion concept in the $P\&S$ method

After the generation of *expanded vector*, the remaining operations are two vector multiplication and partial summation over rows. However, the method requires two sequential kernel launches, one for element-wise multiplication (or production) for the two vectors, which can enjoy fast coalesced memory access. Another one is for carrying out partial summation for each row after the vector multiplication as shown in Fig. 3.7. The second phase, however, can not avoid irregular memory access because of varying length of rows. Also only one thread per row is assigned to perform the addition. To mitigate the

problem, the authors proposed to load the immediate production results into the shared memory via the coalesced memory access. The threads only read from shared memory for the addition operations. But due to the limited resources of shared memory in each streaming multiprocessor (SM) in GPUs, the much slow global memory access will still be needed in case of missing the data in the shared memory.

3.4.2 New parallel SpMV algorithm

Now we present a new parallel sparse matrix-vector algorithm on the GPU-CPU platform, called *segSpMV* method. As we can see, the $P\&S$ method mitigates the irregular memory access for the multiplication phase by using *expanded vector*. However, in the summation phase, it still suffers the irregular memory access issue as the length of rows are irregular. Using shared memory can partially mitigate this problem. However, given the fact that the shared memory is limited, the number of nonzeros per row can't be too large. To see this, let's assume that we need 48 KB data for each block. If we can only have 16KB shared memory (for instance in Tesla T10 GPU). The hit rate (probability of required data in the shared memory) would be only about 33%.

In addition, the $P\&S$ method uses only one thread per row for the summation after loading the data into the shared memory. As a result, the row with less non-zero elements would get the summation result faster compared to the row with more non-zero elements. Therefore, the memory access cannot be full coalesced and the performance of $P\&S$ method is limited by imbalanced workload. Furthermore, its GPU implementation requires two sequential kernel launches. Although it can enjoy memory coalescing in the first

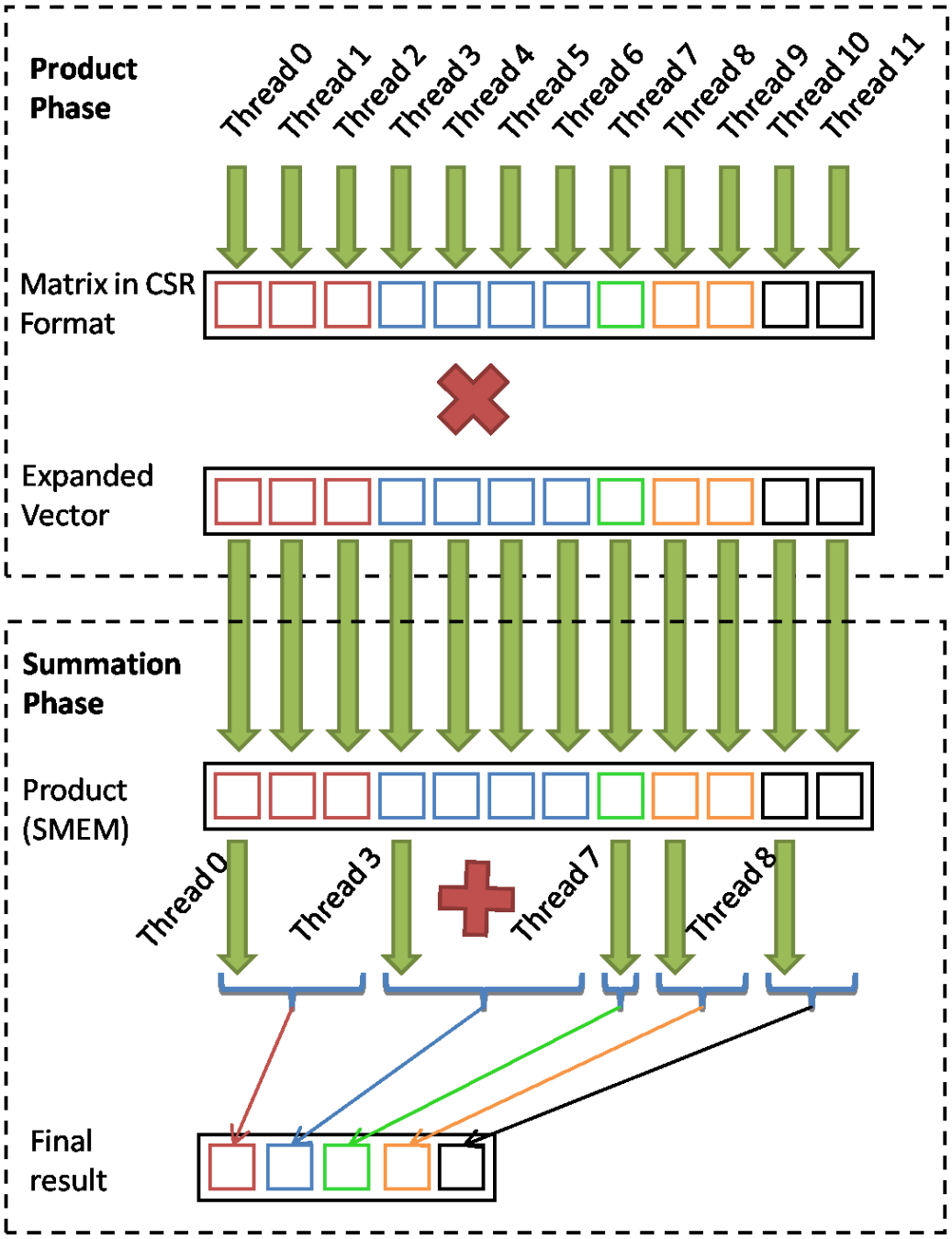


Figure 3.7: The illustration of the $P\&S$ algorithm

product kernel launch, in the second launch for the summation, one has to load intermediate production result vector and the *row_ptr* vector from global memory.

The segSpMV method can overcome the aforementioned problems in the existing $P\&S$ method. The new algorithm is also based on the expanded vector concept for the multiplication phase. But different from the $P\&S$ method, the new algorithm can mitigate the irregular memory access problem in the summation phase, and thus lead to more simple implementation and yet better performance. The main idea is to partition the rows into a number of fixed-length regular segments before the operation. The length of the segment typically is selected to be just bigger than the average number of nonzero elements per row in the given matrix and they also should be the power of 2 for easy reduction operation. For instance, if the average number of nonzero elements is 14, then segment length $2^4 = 16$ is selected. For rows with more nonzeros than the average number, multiple segments will be needed.

After the segment length is determined, each row is partitioned into a number of regular segments. If a segment is not fully filled by the elements from the given row, 0 is padded to the rest of the empty positions in the segment, as shown in Fig. 3.8. In this figure, one 0 is padded at the end of segment2. We perform this segment-based expansion for both original vector and the *expanded vector* of the matrix. After this step, the two segment-expanded vectors are sent to GPU global memory for multiplication and addition phases with just one kernel launch as shown in Fig. 3.8. Note that it takes $O(N_{nz})$ to do the zero padding. In the product phase, each thread first will read two elements from the two segment-expanded vectors respectively via the coalesced memory access from the GPU

global memory. Then each thread multiplies this pair of elements. But it stores the product result immediately into the shared memory instead. In this case, all the intermediate product results from all threads are stored in shared memory, which is ready for the second phase of addition operation right away.

In the summation phase, the new algorithm does not need to check the boundaries of each row any more, which causes the irregular memory access, as it can simply add all the results for each regular segment instead. Because the segment size is fixed, the summation can be very easily done by one thread or by multiple threads via reduction. Also the addition operation will take almost same time for all the threads. We add the *synchronize*() to ensure all the partial results from each segment finish first before they are written back into shared memory using the coalesced memory access. Finally, *segSpMV* adds up the immediate results of segments corresponding to the same row to get the final results in the CPU side, which can be done very efficiently.

We note that the new method will never run out of shared memory, which is the major advantage of the proposed method over the existing approach. The reason is that the amount of memory needed is 4 times of number of threads in each block, as the size of each intermediate element is 4 Byte. So given 1K maximum thread allowed in each block in K20c and K40c GPUs, the maximum memory is just 4KB, which is far less than the 48KB shared memory in each SM. This is also the case for other GPUs as well. As a result, we do not need to write the product results back to global memory and then read them back again, which leads to one more kernel launch. In the addition phase, each thread sums products in one segment and each block is responsible of the same number of segments. The

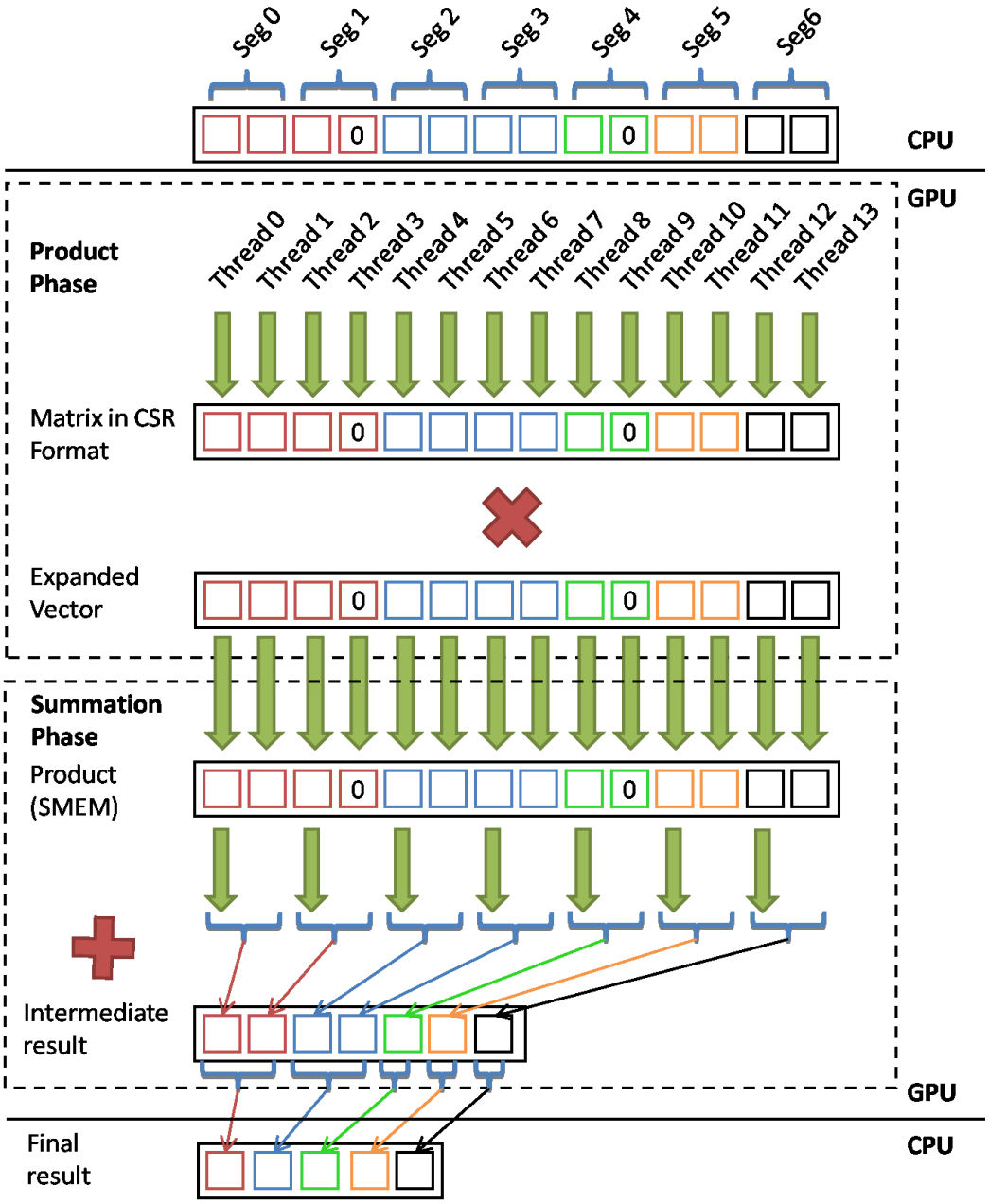


Figure 3.8: The proposed segment-SpMV method or segSpMV method

number of non-zero elements in each row may be different, but all segments are with the same length. Compared to the $P\&S$ method, we do not need to check if a data is cached in

shared memory and do not need to worry about the low hit rate when the number of non-zero elements per row is large as we make full use of the shared memory and has equivalent 100% hit rate in this sense.

3.5 Numerical results and discussions

All the aforementioned methods are implemented in C programming language. The GPU part of the proposed new method is incorporated into the main program with CUDA C programming interface.

To put our new simulator's performance into a right perspective, we compare multi-GPU GMRES with CPU GMRES and a standard LU-based method based on UMFPACK [54]. We remark that we do not compare our multi-GPU GMRES solver with other iterative solvers as most of existing iterative solvers are highly tuned to specific problems, and are not general enough for general linear systems. On the other hand, the proposed multi-GPU GMRES solver is a general solver for any linear dynamic systems, which include but do not limit to the examples of power grid circuits and thermal circuits for both symmetric and non-symmetric matrices. In addition, it does not assume or exploit any structures of the given systems. As a result, it will be more fair to compare our tool with the general LU-based simulator.

These programs are tested on a Linux server with an Intel 2.4 GHz Xeon Quad-Core CPU chip. The host (CPU) side has a total of 60 GBytes memory available. Meanwhile, the server has three GPU cards (devices) as mentioned earlier and are repeated here: one Tesla K40c containing 2880 cores with 12 GBytes global memory, one Tesla K20c con-

taining 2688 cores with 6 GBytes global memory and one Tesla C2075 containing 448 cores with up to 5 GBytes global memory. But we only use the Tesla K40c and K20c in the new multi-GPU GMRES solver.

3.5.1 segSpMV performance comparison on public matrices

To perform the comparisons, several mentioned algorithms have been implemented or obtained from the published sources as listed below:

- segSpMV, the proposed method.
- $P\&S$, the $P\&S$ method.
- $B\&G$-s, the $B\&G$ method using single thread per row [70].
- $B\&G$-w, the $B\&G$ method using one warp per row [70].
- *cu*, the NVIDIA CUSPARSE library SpMV function.

We perform the comparison on the set of matrices from University of Florida Sparse Matrix Collection [55] as shown in Table 3.1 in which *nzsize* means number of nonzeros and *nzperrow* is the average number of nonzeros per row. *seg_length* is the segment length used for the proposed methods. All the matrices are ranked with increasing number of *nzsize* from top to bottom and those matrices represent various matrix structures from wide applications.

Table 3.2 first shows the performance comparison on the matrices in Table 3.1 on the latest Tesla K40c GPU for the five algorithms. It can be seen that the proposed segSpMV method beats all the other algorithms on *ALL* the matrices with various structures. The

Table 3.1: The matrices and their properties from UFL Sparse Matrix Collection

Matrices	row	nzsize	nzperrow	seg_length
scircuit	170998	958936	5.61	8
mac-econ-fwd500	206500	1273389	6.17	8
cop20k-A	121192	1362087	11.24	16
qcd5-4	49152	1916928	39.00	32
cant	62451	2034917	32.58	32
mc2depi	525825	2100225	3.99	4
pdb1HYS	36417	2190591	60.15	64
rma10	46835	2374001	50.69	64
consph	83334	3046907	36.56	32
webbase-1M	1000005	3105536	3.11	4
shipsec1	140874	3977139	28.23	32
pwtk	217918	5926171	27.19	32

Table 3.2: The performance comparison over UFL matrices on K40c GPU

1	2	3	4	5	6	7	8	9	10
Matrices	Algorithm					Speedup			
name	*B&G*-s (ms)	*B&G*-w (ms)	*P&S* (ms)	*cu* (ms)	*seg* (ms)	*B&G*-s	*B&G*-w	*P&S*	*cu*
scircuit	0.352	1.063	0.174	0.195	0.118	2.98	9.01	1.47	1.65
mac-econ-fwd500	0.435	1.757	0.242	0.254	0.153	2.84	11.48	1.58	1.66
cop20k-A	0.932	0.871	0.285	0.251	0.146	6.38	5.97	1.95	1.72
qcd5-4	1.903	0.762	0.523	0.263	0.188	10.12	4.05	2.78	1.40
cant	2.068	0.821	0.528	0.330	0.204	10.14	4.02	2.59	1.62
mc2depi	0.248	2.238	0.297	0.349	0.196	1.27	11.47	1.52	1.78
pdb1HYS	2.416	0.909	0.697	0.373	0.215	11.24	4.23	3.24	1.73
rma10	2.303	1.019	0.702	0.401	0.257	8.96	3.96	2.73	1.56
consph	3.087	1.218	0.791	0.401	0.303	10.19	4.02	2.61	1.32
webbase-1M	14.502	10.843	11.439	1.066	0.531	27.31	20.42	21.54	2.01
shipsec1	3.198	1.704	0.882	0.512	0.378	8.46	4.51	2.33	1.35
pwtk	5.167	2.299	1.256	0.662	0.565	9.15	4.07	2.22	1.17
Average						9.09	7.27	3.88	1.58

average speedups over *B&G*-s, *B&G*-w and *P&S* methods are 9.09×, 7.27× and 3.88× respectively. Speedup in some cases such as *webbase-1M* can be order of magnitude faster over three other algorithms. In addition, we also provide the comparison results between the proposed segSpMV method and NVIDIA CUSPARSE library function. The speedup ranges from 1.17× to 2.01×, with average 1.58×. Although the speedup highly depends on the benchmark matrices, we see the $> 1\times$ speedup on all the cases.

3.5.2 Multi-GPU segSpMV implementation and performance comparison

To further utilize the multiple GPU resources and make the proposed segSpMV method more scalable for handling much larger problems, we further extended segSpMV algorithm into the multi-GPU platforms.

Specifically, the segSpMV method can be easily divided into several tasks. First, we partition the two expanded vectors into several segment groups, and each group is managed by a CPU thread. The number of groups can be determined by the number of GPU devices on the server. Second, each CPU thread passes the corresponding segments to one GPU device, and GPU just finishes the computation of multiplication and addition phases with one kernel launch. Since the sparse matrix and vector are already expanded into several segments with a fixed length, the task partition and distribution become very simple. Furthermore, the segSpMV method is very multi-GPU friendly as there is no inter-GPU communication. Each GPU can still enjoy the full coalesced memory access and shared memory utilization.

Our multi-GPU server consists of one Tesla K40c, one Tesla K20c, and one C2075 GPUs. The server also consists of two 8-Core Xeon E5-2670 CPUs, DDR3-1600 64GB memory. The Tesla K40c and K20c GPU are built on the NVIDIA Kepler compute architecture and have 2880 and 2688 CUDA parallel processing cores respectively. The K40c is capable of running 4.29 Tflops per second of single precision processing performance while K20c has the peak 3.95 Tflops single precision floating performance. C2075 is based on previous Fermi architecture GPU with 448 cores and 1 Tflops peak single precision performance.

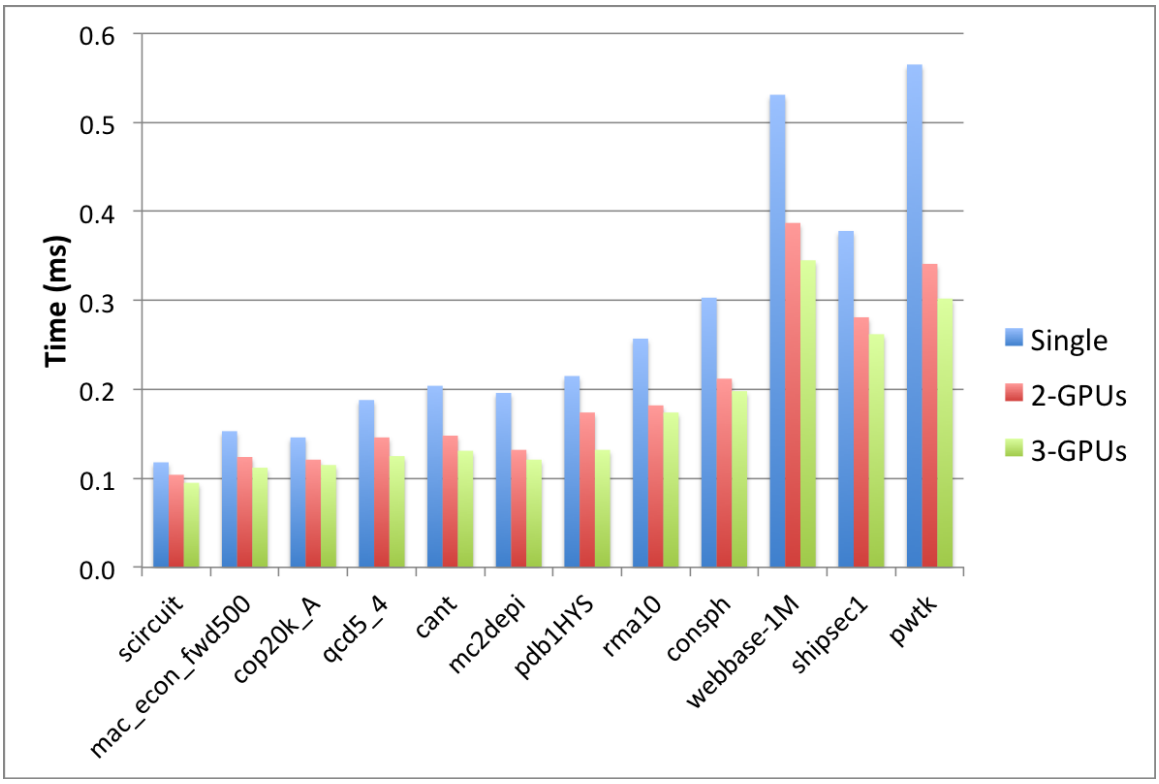


Figure 3.9: The performance comparison of multi-GPU segSpMV method

The resulting *multi-GPU segSpMV* method can gain further speedup as shown in Fig. 3.9 in addition to the added scalability. The performance comparison is based on the matrices in Table 3.1 for single GPU (K40c), 2-GPUs (K40c and K20c) and 3-GPUs (K40c, K20c and C2075). It can be seen the performance differences are very small when the matrix size is small. It is due to the overhead of creating new CPU threads, starting GPU and performing synchronization. However, the speedups in larger cases are much better. For example, for the largest matrix *pwtk*, the 2-GPUs and 3-GPUs implementations are 66% and 87% faster than the single GPU implementation. We also notice that the 2-GPUs and 3-GPUs implementations have similar performance. We also notice that the K40c and K20c

are much more powerful than the C2075. So the computing speed of 3-GPUs implementation is mainly determined by C2075, which limits the performance improvement. But the results from Fig. 3.9 clearly demonstrates the advantages and benefits of the proposed multi-GPU segSpMV over the single GPU segSpMV method.

3.5.3 Accuracy comparison and discussions

We first test the accuracy and efficiency of our solver on the power grid circuits from IBM benchmark suite [71]. There are 6 benchmark circuits with sizes ranging from forty thousand to three million nodes in the interconnection. The information of these benchmarks can be retrieved from their website. We show the matrix sizes of their circuit MNA models in Table 3.3. Also in the same table, the running time spent in LU factorizations and LU solves of the backward Euler equations are also listed. The equation solved here is stated in Eq. (3.3). Since we use uniform discretization in the time domain, the time step length h remains the same on all the steps. In addition, all of our examples are linear circuits, and the matrices $\mathbf{G}$ and $\mathbf{C}$ do not change either. As a result, the LU factorization only needs to be calculated once on $\mathbf{G} + (1/h)\mathbf{C}$ and its triangular $\mathbf{L}$ and $\mathbf{U}$ are reused for all the transient steps. The time measurements in Column "LU fact." are the one time cost of LU factorization, and those in Column "LU solve" are time spent on LU triangular solve on one time step, i.e., solving $\mathbf{Ax} = \mathbf{b}$ with reuse of LU factors.

The error tolerance of all of our GMRES solvers is set to 10^{-7}. A smaller tolerance guarantees higher accuracy, but also leads to more iterations and longer solving time. During our extensive experiments with the benchmarks, we have found a 10^{-6} tolerance is good and accurate for most cases. Nonetheless, we use 10^{-7} for all experiments as this will give us

statistics according to the same standard. We do not push our tool only for a demonstration of speed with the sacrifice of accuracy.

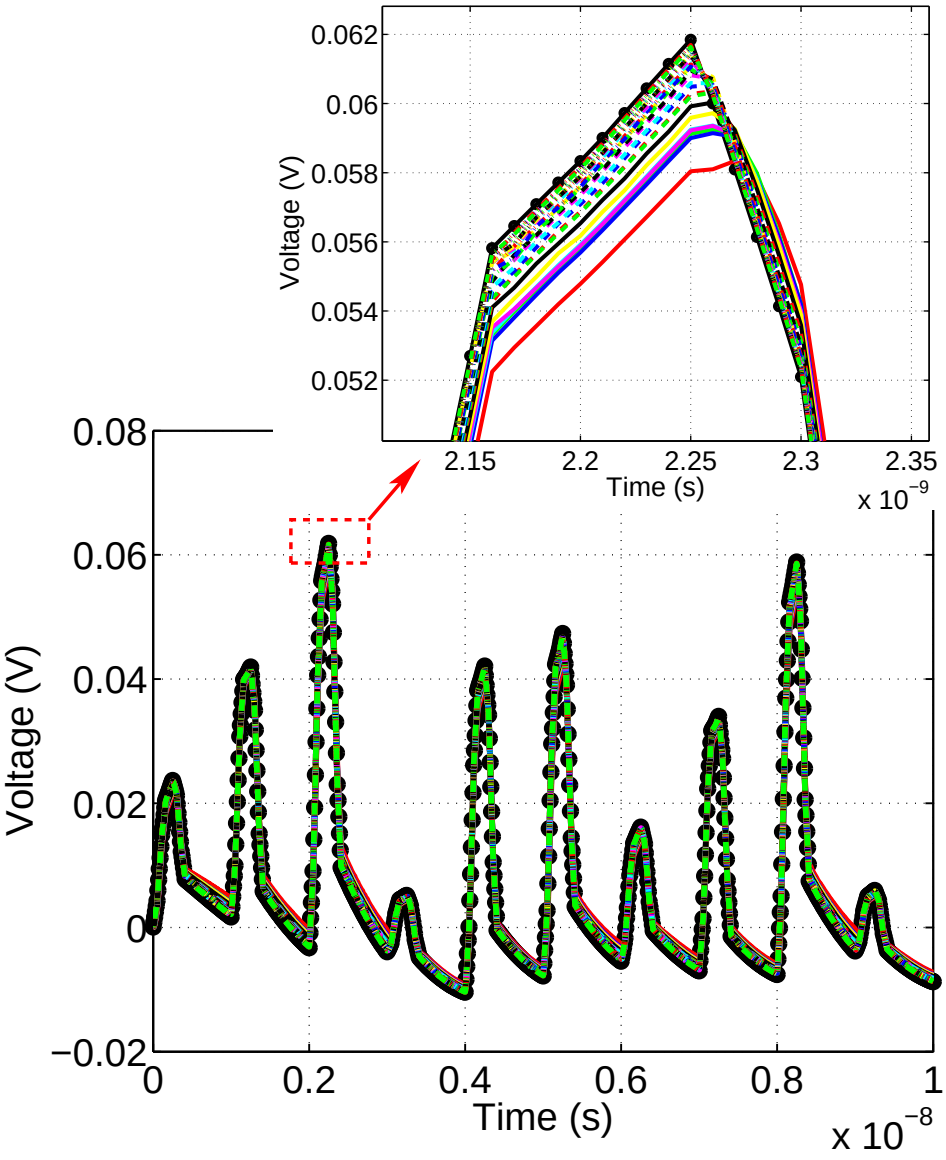


Figure 3.10: Transient waveforms of LU and GPU GMRES at port node n0_5480720_1102640 in ibmpg6t. The black curve with dots is from LU direct method. All other colored curves are results of GMRES with preconditioners set to different ILU threshold, i.e., from 0.1 to 3.0.

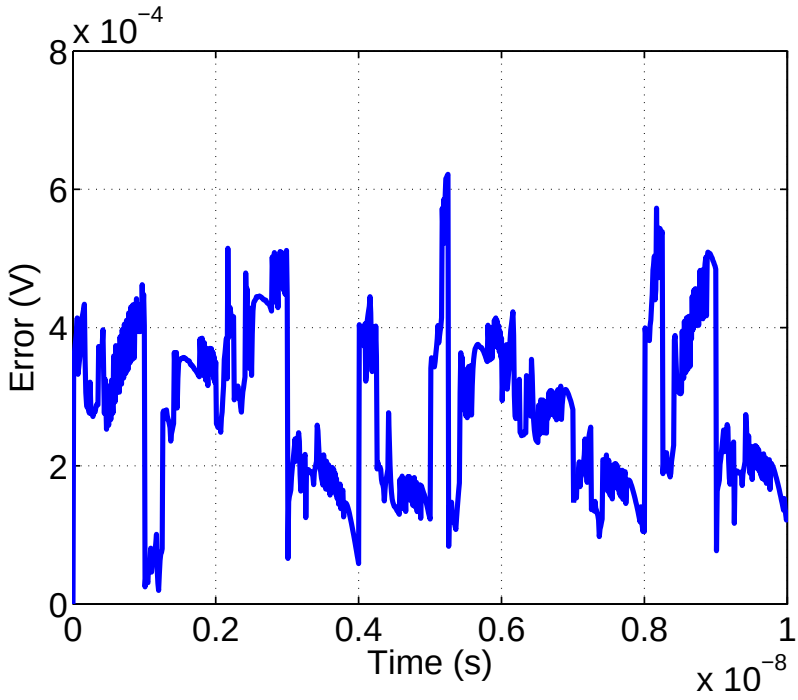


Figure 3.11: The error of GPU GMRES result compared to LU golden result. This curve is calculated at node n0_5480720_1102640 of ibmpg6t, whose waveform is shown in Fig. 3.10.

Fig. 3.10 shows the simulation results of a benchmark circuit ibmpg6t, from IBM. It is a voltage waveform at node n0_2679_17913. We plot the waveforms of the direct LU method and multi-GPU GMRES with preconditioner on the same figure, and the accuracy of GMRES result is quite satisfactory since the two curves are closely overlapped. To further show the accuracy, we plot the errors of the GMRES curve, i.e., the difference between GMRES result and LU result, in Fig. 3.11, which shows about 1% maximum relative error. We have verified all the examples, especially waveforms at the observation port nodes listed by `.print` command in IBM netlists, and all the waveforms from GPU GMRES agree with the LU golden results.

Table 3.3: Statistics of IBM power gird benchmarks and solver performance. Column 14 lists the speed up of GPU GMRES over LU method on all the 1,000 time step points in a transient simulation calculated as $\frac{C3+1000 \cdot C4}{C6+C9+1000 \cdot C12}$.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
		LU		GMRES								sp. up DC	sp. up
circuit	matrix	fact.	solve	ILU	precond.	solving on DC			solving on tran. step (ave.)			$\frac{C3+C4}{C6+C9}$	
name	size	(s)	(s)	thres.	setup (s)	# iter	CPU (s)	GPU (s)	# iter	CPU (s)	GPU (s)		
ibm1t	54,265	0.19	0.02	2.1	0.10	33	0.35	0.07	7	0.03	0.01	1.2	2.0
ibm2t	164,897	9.93	0.06	1.2	0.62	143	3.50	0.51	23	0.54	0.06	8.8	1.2
ibm3t	1,043,444	638.7	0.87	2.6	5.03	25	6.41	1.55	6	1.10	0.35	97	4.2
ibm4t	1,214,288	904.7	1.01	1.9	9.65	77	23.2	4.69	10	3.15	0.57	63	3.3
ibm5t	2,092,148	241.6	0.60	1.5	5.80	118	22.1	4.41	17	3.36	0.49	24	1.7
ibm6t	3,203,802	174.3	0.82	2.2	12.49	42	15.2	3.44	9	3.40	0.52	11	1.9
rlc80	32,064	6.97	0.01	1.8	0.12	29	0.12	0.28	4	0.01	0.003	17	5
rlc100	50,200	28.60	0.02	1.8	0.17	32	0.18	0.38	4	0.02	0.003	52	14
rlc120	72,384	102.2	0.05	1.9	0.26	32	0.28	0.44	4	0.02	0.008	146	18
rlc140	98,616	255.6	0.08	2.0	0.36	32	0.39	0.49	4	0.04	0.008	301	38
rlc160	128,896	726.3	0.15	2.0	0.48	34	0.51	0.52	4	0.06	0.008	726	97
rlc180	163,224	2,033.6	0.28	2.0	0.68	34	0.65	0.63	4	0.10	0.008	1552	248
rlc200	201,600	4,191.3	0.39	2.0	0.85	35	0.82	0.64	4	0.14	0.025	2813	173
rlc220	244,024	6,750.9	0.54	2.1	1.09	35	1.01	0.78	4	0.19	0.017	3610	386
rlc800	3,235,200	-	-	2.1	25.93	35	16.86	0.63	5	1.56	0.13	-	-
rlc1000	5,056,000	-	-	2.0	91.88	36	28.09	0.75	6	3.66	0.22	-	-
rlc1200	7,281,600	-	-	2.2	139.06	36	46.43	1.39	5	5.12	0.33	-	-

3.5.4 Computing time comparison and discussions

Table 3.3 lists the running time measurements in all the benchmarks, including IBM examples and self-generated rlc grids. The Column 5 (C5) gives the threshold value used for control the fill-ins in the ILU preconditioner. The C6 lists the preconditioner setup time, the C9 is for the multi-GPU GMRES solving time without initial guess available, and the C12 is for the multi-GPU GMRES solving time on each transient point, when good initial guess is available. The speedup of multi-GPU GMRES over LU on DC solving is

listed in the C13. The speedup of multi-GPU GMRES over LU on the whole simulation (1000 time steps) is listed in the C14.

We first discuss the results on the IBM examples. Among the six IBM circuits, multi-GPU GMRES brings reasonable speedup over LU factorization. To make a fair competition with LU, the speedup on DC solving, i.e., the first GMRES solve without any good initial guess available, shall be calculated as (C3 + C4)/(C6 + C9). The biggest speedup for this initial DC solving is 97× , which happens in the case of ibmpg3t. We notice that the speedup does not always go up with the size of the circuit as shown in Table 3.3. We observe that these IBM benchmarks vary not just in sizes, but also in the circuit structure and thus the their matrix structures. But still the proposed parallel GMRES solver show decent speedup over the direct method on these industrial design examples. We also observe that the multi-GPU GMRES solver will have about 4-5× speedup over their CPU version of GMRES solver on those IBM benchmark circuits (not shown in the table), which clearly shows the advantages and benefits of GPU based computing.

For transient analysis, we observe that when the LU factors are available, it seems cheaper for LU triangular solve than iterative methods to compute the solution. Since fixed time step is used in our simulator and the triangular LU factor matrices do not change as we mentioned in the previous sections, it is very understandable that GMRES does not superbly beat the triangular solve if the examples are relatively small. However, as the average running time listed in C12 of GMRES solve is smaller than C4 of LU solve, the total reduction of cost will still be favored when there are a lot of transient steps. If LU factorization has to be done many times, as happens in transient simulation with changing

time steps, GMRES solver will be faster than the LU factorization. Nowadays, power grid simulation also needs to consider nonlinear effects and the MNA matrix is updated during the transient simulation. With more running time spent on many factorizations, LU method will definitely be outperformed by the GPU GMRES solver.

Now we discuss the results on some RLC mesh circuits, which are the middle eight examples in Table 3.3 with "rlc" in circuit names. Those power grid networks are generated based on RLC mesh grid circuit model shown in Fig. 3.1. We observe that the speedups of the proposed method over LU factorizations in both DC and transient analysis is much larger (ranging from 5 to 3610) and speedup goes up with the sizes of the circuits. This indicates that the structures of the power grid networks has huge impacts on the solving efficiency and their final computing speed. Similarly, we observe that the multi-GPU GMRES solver will have about 3-12× speedup over their CPU version of GMRES solver on those IBM benchmark circuits for transient analysis, although the speedup is marginal for DC analysis. As a result, it seems that IBM examples favor the LU based solver, while our mesh-structured RLC networks favor the proposed GMRES solver.

To show the added scalability of the new parallel GMRES solver on multi-GPU platforms, we also provide three very large RLC mesh circuits, which are the rlc800, rlc1000 and rlc1200. They are all million-sized circuits and cannot be handled in single GPU card with limited global memory. But our multi-GPU GMRES solver is able to handle such large circuits easily. We notice that the LU factorization method is too slow for these large circuits. As a result, we don't show the results and speedup comparison for the LU solver. We observe that the multi-GPU GMRES solver will have about 12-17× speedup

over their CPU version of GMRES solver on those large circuits for transient analysis, and the speedup for DC analysis is also between 30%-60%, which is better than the speedup on small circuits.

3.5.5 Preconditioner study and discussions

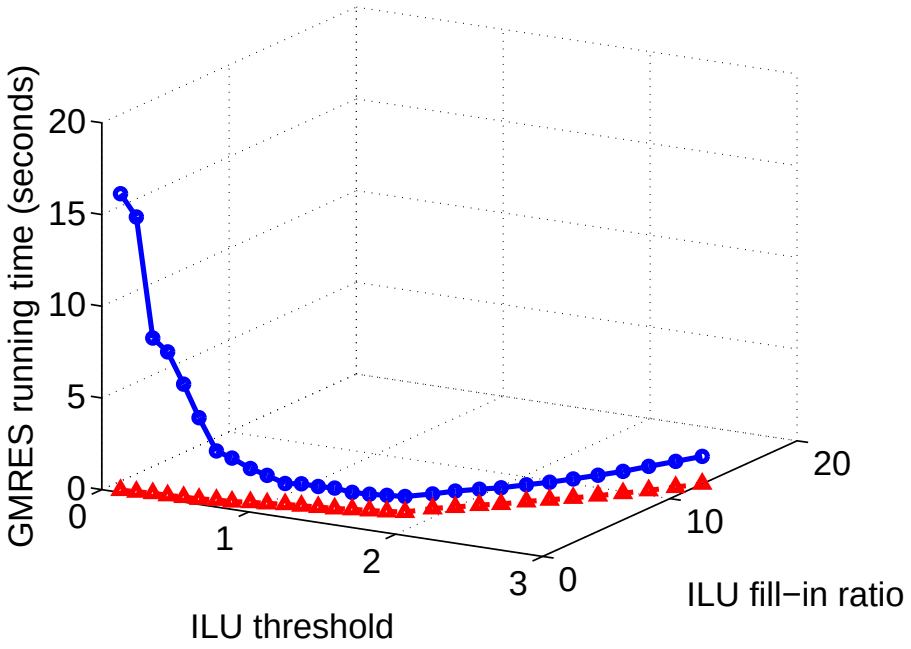


Figure 3.12: The impact of ILU threshold on fill-in ratio and GMRES solving time. The blue curve in 3D space is GMRES solving time with respect to threshold and fill-in ratio, and the red curve on the bottom plane reflects the changes of fill-in ratio caused by different threshold values. All the measurements are from ibmpg4t.

Now, let us study the quality of an ILU preconditioner. The fill-in ratio is a good indicator about the quality of incomplete LU preconditioner. It is calculated as the ratio of the number of fill-in elements in incomplete LU factors $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ over the number of

Table 3.4: The performance comparison of ILU preconditioners with different fill-in ratios. The same circuit matrix from IBM power gird benchmark ibmpg4t is used in all the cases.. GMRES convergence tolerance is set to 10^{-7}.

threshold	precond setup (s)	ILU fill-in	# iter on DC	# iter per tran step	total time (s)
0.1	5.46	0.31	3447	913	12531.6
0.3	5.69	0.53	1469	440	6413.9
0.5	5.28	0.65	690	310	4773.2
0.7	5.93	0.92	480	115	1905.8
0.9	6.18	1.29	366	68	1358.9
1.1	6.67	1.70	237	32	812.6
1.3	6.92	1.99	210	26	821.8
1.5	7.28	2.35	126	19	720.5
1.7	7.74	2.77	109	16	664.3
1.9	9.65	4.06	77	10	645.6
2.1	12.38	5.42	47	7	727.5
2.3	16.05	6.81	39	6	753.3
2.5	20.83	8.18	30	5	804.8
2.7	27.57	9.78	37	5	941.3
2.9	37.30	11.61	37	4	1132.7
3.0	42.68	12.55	19	4	1233.4

non-zero elements in the original coefficient matrix $\mathbf{A}$, i.e,

$$\text{fill-in ratio} = [\text{nnz}(\tilde{\mathbf{L}}) + \text{nnz}(\tilde{\mathbf{U}}) - n]/\text{nnz}(\mathbf{A}).$$

Notice that the diagonal of lower triangular factor nnz($\tilde{\mathbf{L}}$) is unitary and need not be stored in practice. This also explains the subtraction of matrix size n in the equation above. For the simplest incomplete LU preconditioner ILU0, which computes the LU factorization but drops any fill-in elements in $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ outside of the nonzero pattern of $\mathbf{A}$, the fill-in ratio is 1.0. This means the number of non-zero elements in ILU0 factors are equal to that of $\mathbf{A}$'s. To the best of our knowledge, NVIDIA has released a function of ILU0 factorization in the most recent CUSPARSE 5.0 version [67]. However, it has no fill-ins and does not support row/column permutation, and our experiments show that the two limitations hurt its applicability to the circuit cases here. Instead, we use the ILU package from [72], who allows different fill-in ratios by modifying the dropping threshold. This threshold parameter controls the dropping rule during incomplete LU factorization and affects the behavior of ILU preconditioner. The detailed description of the dropping rule can be found in [73]. Though low fill-in ratio implies a simple structure in the two triangular factors and a possibly faster computation in GPU's triangular solve, it results in more iterations in GMRES solver and may not be optimal in terms of overall computation time of GMRES. In addition, the time spent on preconditioner construction also grows up in order to compute more fill-in elements. Table 3.4 shows the relationship among the threshold, fill-in ratio, the iteration numbers, and the total GMRES CPU time. It can be seen that the CPU time reaches the minimum value when the threshold is 1.9. Fig. 3.12 depicts the aforementioned relationships. The data in this figure are measured from 30 runs of the same circuit *ibmpg4t*, where only the threshold is changed from 0.1 to 3.0 with 0.1 increment. The effects of this change on fill-in ratio and GMRES time on each time step are shown by two curves.

3.6 Summary

In this chapter, we have proposed an efficient parallel solver *GPU-GMRES* for large linear dynamic systems. The new solver is based on the preconditioned GMRES solver implemented CPU-GPU platforms. The proposed GPU-GMRES solver is based on the very general and robust incomplete LU based preconditioner. We have shown that by properly selecting the right amount of fill-ins in the LU factors, a good trade-off between GPU efficiency and convergence rate can be achieved for the overall best performance. In addition, a new fast parallel SpMV multiplication algorithm is proposed to further accelerate the GMRES solver. The new algorithm, called *segSpMV*, can enjoy full coalesced memory access. To further improve the scalability and efficiency, *segSpMV* method is further extended to multi-GPU platforms. The resulting *multi-GPU segSpMV* can deliver further performance enhancement for the resulting *multi-GPU-GMRES* solver. Furthermore, we have properly partitioned the major computing tasks in GMRES solver to minimize the data traffic between CPU and GPU, which further boosts performance of the proposed method. Experimental results on the set of the published IBM benchmark circuits and mesh-structured power grid networks have shown that the GPU-GMRES solver can deliver order of magnitudes speedup over one direct LU solver. The resulting *multi-GPU-GMRES* can also deliver 3-12× speedup over the CPU implementation of the same GMRES method on transient analysis.

Chapter 4

EM-Based on-Chip Aging Sensor for Detection and Prevention of Recycled ICs

4.1 Introduction

The counterfeiting and recycling of integrated circuits (ICs) have become major problems in recent years, potentially impacting the security of electronic systems especially for military, aerospace, medical and other critical applications. In addition to diminishing system dependability and usability, counterfeiting reduces total revenue of companies from their research and development efforts, discourages innovation through the theft of intellectual properties (IPs), and produces low-quality products under established brand names [37]. A counterfeit component is defined as an electronic part that is not genuine because it is an

unauthorized copy; it does not conform to the original component manufacturer's (OCM) design, model, and/or performance; or it is not produced by the original component manufacturer or is produced by unauthorized contractors; it is an off-specification, defective, or used OCM product sold as "new" or working; it has incorrect or false markings and/or documentation [38].

Today the most widely reported type of counterfeit parts is the recycled type. It is reported that in today's supply chain, more than 80% of the counterfeit components are recycled [39]. These used or defective ICs enter the market when electronic "recyclers" divert scrapped circuit boards away from their designated place of disposal for the purposes of removing and reselling the ICs on those boards. The recycling process involves removing ICs from the board or even dies in the ICs. There are several security issues associated with these ICs. Firstly a used IC can act as a ticking time bomb [40] since it does not meet the specification of the OCM of the ICs; secondly additional die on top of the recovered die can carry a back-door attack, sabotage circuit functionality under certain conditions, or cause a denial of service [41].

The detection methods for recycled chips can be classified into physical methods and electrical methods [37]. Physical methods consist of incoming inspection methods such as visual inspection, X-ray imaging, package analysis method such as laser scanning microscopy, delid method, and the material analysis method such as using Fourier transform infrared, and X-ray fluorescence. Electrical methods contain the parameter tests, function tests, built-in tests and structural tests. In general, physical methods can be applied to all part types, but some of the methods are destructive and take hours to test. As a result,

sampling is required to certify a batch of parts by observing a small number of parts. On the other hand, conventional electrical test methods are non-destructive and time efficient, yet they can be very expensive because such techniques are not necessarily designed for counterfeit detection. Electrical test techniques are advantageous because the sampling is not required, and all parts can be tested. However, there are some issues associated with electrical tests that must be addressed.

In order to fast detect and effectively prevent the recycled chip, one viable approach is to insert a lightweight aging detecting sensor, which can directly tell the usage of the chips and some early efforts have been explored [42, 43, 44]. Method in [43] designed the ring-oscillator-based (OR-based) aging sensor that relies on the aging effects of MOSFETs to change a ring oscillator frequency in comparison with the reference one embedded in the chip. As the chip ages owing to the wear-out mechanisms such as negative biased temperature instability (NBTI) and hot carrier injection (HCI), the shift threshold voltage of MOSFET devices, thus the frequency of ring oscillator indicates the level of aging, and provides a simple readout of the value. However, this method can only give very rough estimation of the usage age of the chip as the shift of the frequency depends on many factors. In order to mitigate this problem, the antifuse-based (AF-based) sensor was developed in [37]. The AF-based sensor essentially is a counter, which counts the clocks or derivatives of the clock events to log the usage of the chip. The antifuse memory is used to make sure the data in the count will not be erased or altered by attackers. However, the AF-based sensors suffer large area overhead especially when more accurate usage is required [37]. Another problem with this method is that it may not reflect the true aging-dependent usage of a chip.

For instance, it will log the same usage time for a chip for different on-chip temperatures, however, which can have dramatically impacts on the aging effects from electromigration, NBTI and HCI [45].

In this chapter, we propose a new lightweight on-chip aging sensor, which is based on the electromigration-induced aging effects for fast detection and prevention of recycled ICs. Instead of using traditional aging effects from devices (such as MOSFETs), the new EM-based aging sensor exploits the natural aging/failure mechanism of interconnect wires to time the aging of the chip. As a result, comparing with existing the ring-oscillator-based aging sensor, it has following two advantages: first, this structure is much simpler as it only requires some metal interconnect wires, which will be driven by DC currents. In comparison, the ring oscillator has to detect the threshold voltage shift first, which is more difficult, and then estimate the aging of the chip. Second, it is more accurate as we can measure the EM-induced failure (such as wire resistance changes) time with more accurate than the frequency shift over time. The new sensor is based on a newly proposed hydrostatic stress evolution model of EM effects for accurate prediction of the EM failure [46]. As a result, we can design the interconnect wire structures based on the copper interconnect technology so that the resulting wires can have detectable EM failure at a specific time with sufficient accuracy. In order to mitigate the problem of the inherent variations in the metal grain sizes and assess its impacts on the nucleation time of metal wires, a number of parallel properly structured wires are employed in the sensor. The parameters of the wires are optimized with using the new EM model. Our experimental results show that the proposed aging sensor can accurately predict the targeted failure times in the presence of both inherent

uncertainties. Our study also shows that more parallel wires will lead to more accurate statistical predictions at costs of more areas.

This chapter is organized as follows. Section 4.2 reviews the EM effect and recently proposed physics-based EM model. In Section 4.3, we present the new lightweight on-chip aging sensor circuit as well as the interconnect wire structures. Several statistical and variational analyses are presented in Section 4.4. Last, Section 4.5 concludes.

4.2 Review of EM effects and EM models

4.2.1 Review of EM-induced failure effects

The proposed on-chip aging sensor is based on the observation that the EM-induced failure of interconnect wires can be designed such that the wires can fail at a specific time frame detected by the increase of their resistances over a pre-defined threshold. To understand this, let's first have a brief review of the EM failure effects from the first principles and then we present the problems and solutions to design the wire structure for timed failure based on EM physics.

EM is a physical phenomenon of the migration of metal atoms along a direction of the applied electrical field. Atoms (either lattice atoms or defects/impurities) migrate toward the anode end of the metal wire along the trajectory of conducting electrons. This oriented atomic flow, which is caused mostly by the momentum exchange between atoms and the conducting electrons, results in metal density depletion at the cathode, and a corresponding metal accumulation at the anode ends of the metal wire. This depletion and accumulation happen because atoms cannot easily escape the metal volume.

The rate of EM, as defined by the Nernst-Einstein equation, depends on the atomic diffusivity, meaning different materials are characterized by different rates of EM. Typical interconnect metals, such as copper (Cu) and aluminum (Al), are prone to EM, due to their high self-diffusivity. Refractive metals, such as tungsten, tantalum, and titanium, demonstrate strong resistance to EM.

The interconnect segment here means a continuously connected, highly conductive metal within one layer of metallization, terminated by diffusion barriers. Thin layers of refractive metals form these diffusion barriers for Cu atoms, preventing them from diffusing into inter-layer (ILD) and inter-metal dielectrics (IMD). When metal wire is embedded into a rigid confinement, which is the case with interconnect metallization, the wire volume changes (induced by the atom depletion and accumulation due to migration) creating tension at the cathode end and compression at the anode ends of the line. Over time, the lasting unidirectional electrical load increases these stresses, as well as the stress gradient along the metal line. Fig. 4.1(d) shows the stress evolution in a straight wire over time. The cathode node has the tensile stress (positive stress) built up, while the anode node has compressive stress generated (negative stress). In some cases, usually when a line is long, this stress can reach a critical level, resulting in void nucleation at the cathode and/or hillock formation at the anode end of line as shown in Fig. 4.1(d). Fig. 4.1(a) shows what the actual void and hillock look like. Different physical mechanisms can be responsible for generating these damages. In the case of voiding, existing cohesive or interfacial micro-cracks near or at the barrier/Cu interfaces can develop into a void by action of the appropriate stresses. Hillock formation, which is a compression-induced extrusion of metal into the surrounding dielectric

that can cause a shortage between neighboring metal lines, can be initiated by micro-cracks in the adhesion/barrier layers. However, typically the voids are the major defects from EM.

In addition to voids nucleated at the cathode end of line, where a divergence in atomic flux happens (atom flux is terminated at the barrier interface), many voids are nucleated down to the polycrystalline metal line toward the anode end at any location characterized by the atom flux divergence. These are the triple points formed by intersections between grain boundaries (GB) and the top dielectric barrier (typically composed of SiCN), or contacts between three neighbor grains (Fig. 4.1(b)). It is well known that atoms diffuse much faster along GB and interfaces than through the grain interiors, making GBs and interfaces the major venues for EM. Those triple points where the number of outward diffusion channels exceeds the number of inward channels can develop depletion in metal density, leading to possible void nucleation. Nucleated voids, depending on the local texture of neighboring grains, can grow in size or disappear. As shown in Fig. 4.1(c), two major mechanisms of void growth are: (i) scavenging the vacancies that migrate to the void due to the stress gradient between the void surfaces (zero stress) and the surrounding metal (tensile stress), (ii) agglomeration of voids travelling along the metal line toward the cathode end (against the electron flow) due to the capillarity effect. GBs with different crystallographic orientations are characterized by different atomic diffusivities, governed by a variation in grain crystallography. This variation, together with a random distribution of grain sizes inside metal lines, show that why identical metal lines, characterized by same geometries and same electrical load, demonstrate different time-to-failure (TTF). This TTF represents the instant in time when an increase in line electrical resistance caused by the void growth

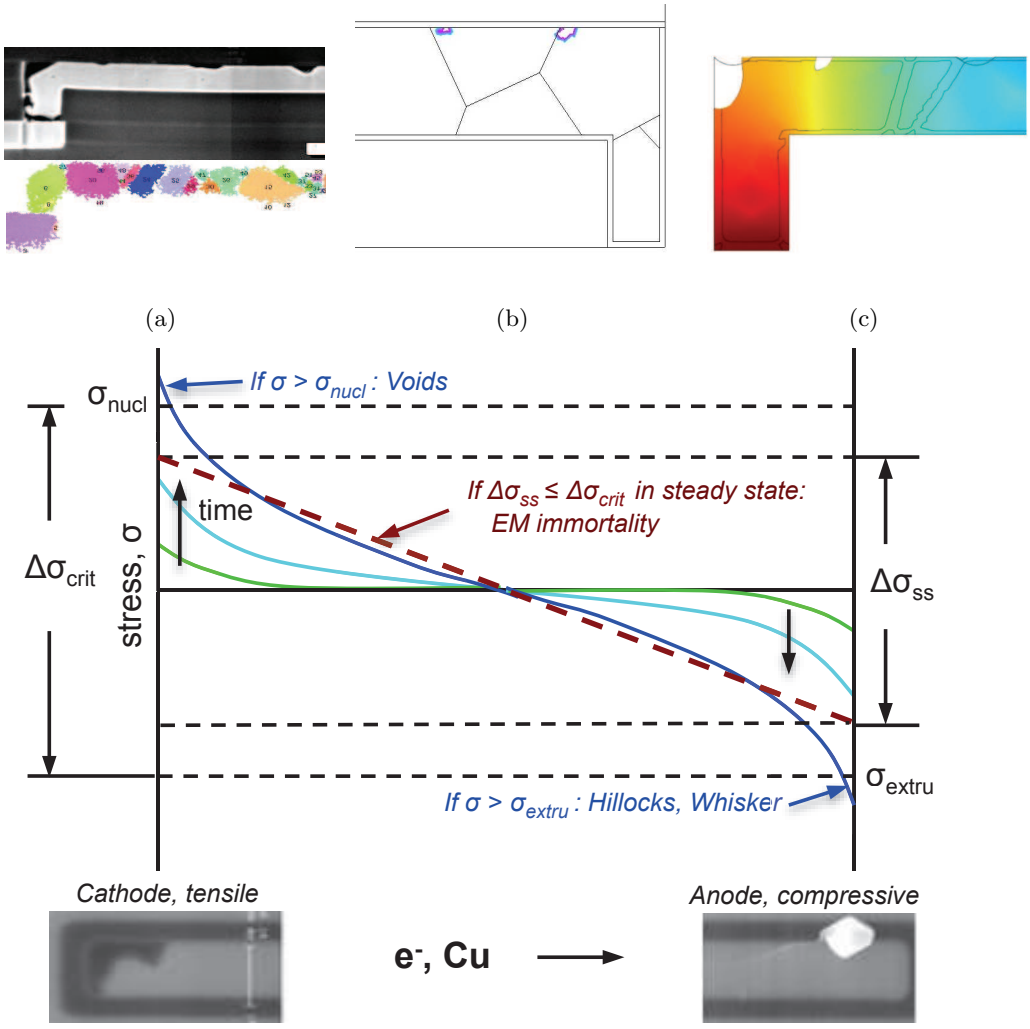


Figure 4.1: (a): TEM picture of voids nucleated at the top interface, [1], (b) and (c): simulated kinetics of the void nucleation at the triple points and growth (electron flow from right to left), [2], (c): simulated growth of the line corner void by scavenging the vacancy flux and agglomerating with the small voids drifting along the top interface [3]. (d) The EM-induced stress development and distribution of an interconnect wire.

reaches a critical level (for example, a 10% increase over the original value). This inherent uncertainties in TTF is one of the challenges to design accurate aging sensor. Traditionally the EM effects are modeled by the Black's equation:

$$MTTF = Aj^{-n}exp\{E_a/kT\} \tag{4.1}$$

which calculates the segment MTTF based on known current densities (j) and temperatures (T), is the subject of growing criticism. Here, k is the Boltzmann's constant, E_a is the EM activation energy. The symbol A is a constant, which depends on a number of factors, including grain size, line structure and geometry, current density, thermal history, etc. Black has determined the value of n as equals to 2. However, it is a todays common understanding that n depends on residual stress and temperature [74], and its value is highly controversial. In addition, as it was shown in a number of experiments, see for example [74], E_a is a function of the current density. All these observations make rather controversial the widely accepted methodology of calculating the MTTF at use condition, represented by chip operational current density and temperature, while using n and E_a determined at the stressed (accelerated) condition, characterized by high current densities and elevated temperatures.

For the proposed aging sensor design, we need much more accurate and physics-based EM model to estimate the failure times, which is critical to ensure the accuracy of the aging sensor.

4.2.2 Physics-based EM model

In the following, we review the recently proposed physics-based EM model for accurate estimation of nucleation time and resistance change rate of a wire. We start to review the well-known partial differential equation, which governs the stress evolution process. It was first proposed by Korhonen [75] and further developed by other researchers; see for example [76, 77]. Since the atomic flux divergence results in the volumetric strain, it is easy to derive the one-dimensional diffusion-like equation for the hydrostatic stress field $\sigma(x,t)$ [75]:

$$\frac{\partial \sigma}{\partial t} = \frac{\partial}{\partial x}\left[\kappa\left(\frac{\partial \sigma}{\partial x} + \frac{eZ\rho j}{\Omega}\right)\right] \tag{4.2}$$

Here, $\kappa = D_a B\Omega/kT$, where D_a is the atomic diffusivity, and B is the bulk modulus. Solution of this initial-boundary value problem is the infinite series [75].

Approximate value of void nucleation time (t_{nuc}) extracted from this solution, which is determined as an instant in time when stress at the cathode end of the line ($x = -l/2$) reaches σ_{crit}, corresponds well to an analytical formulation of t_{nuc} derived from the approximate solution of continuity equations for evolution of vacancy and plated atom concentrations (see, for example [3]) in the confined 1D line [78]:

$$t_{nuc} \approx \tau^* e^{\frac{E_V}{kT}} e^{-\frac{f\Omega}{kT}\left(\sigma_{Res} + \frac{eZ\rho l}{4\Omega} j\right)} ln\left\{\frac{\frac{eZ\rho l}{4\Omega} j}{\sigma_{Res} + \frac{eZ\rho l}{4\Omega} j - \sigma_{crit}}\right\} \tag{4.3}$$

where $\tau^* = \frac{l^2}{D_0} e^{\frac{E_D}{kT}} \frac{kT}{\Omega B}$. Here, E_V and E_D are the activation energy of vacancy formation and diffusion, f is the ratio of volumes occupied by vacancy and lattice atom. In this model, we consider the residual stress of $\sigma_{Res} = \sigma_T + (B/9)(R/\delta)exp\{-E_V/kT_{ZS}\}$ when electrical stressing was applied. Here, σ_T is the thermal stress developed in the metal line

confined in the ILD/IMD dielectric during cooling from the zero stress temperature T_{ZS} down to the temperature of use condition, $(B/9)(R/\delta)exp\{-E_V/kT_{ZS}\}$ is an additional stress generated by vacancy relaxation to the equilibrium concentration corresponding to new stress value and temperature [78], R is the mean grain size, and δ is the GB thickness. Dependence of t_{nuc} on grain size allows one to introduce a simple statistical model for void nucleation at the line cathode edge. Note that in the new physics-based EM model, one needs to explicitly consider the residual stresses, σ_{Res}, which can have huge impacts on the nucleation time and thus the failure time of a wire. As a result, it is important to have an accurate estimation of residual stresses and more accurate residual stress can be computed using multi-scale numerical method [79].

The second problem we consider in the proposed EM model is how the wire resistance change over time once void nucleation happens. We need to figure out how the void grows and the growth rate, as well as the related resistance change rate. As we know, voids are formed at t_{nuc} and grow at $t > t_{nuc}$. The wire resistance starts to increase over time in the growth phase. Since the drift velocity of the void edge relates to atomic flux as $\vartheta = \Omega j$ [80], we can express it as: $\vartheta = \frac{D}{kT} eZ\rho j$. Kinetics of the wire resistance change can be approximately described as:

$$\Delta r(t) = \vartheta(t - t_{nuc}) \left[\frac{\rho_{Ta}}{h_{Ta}} \left(\frac{1}{2H} + \frac{1}{W} \right) - \frac{\rho_{Cu}}{HW} \right] \tag{4.4}$$

Here ρ_{Ta} and ρ_{Cu} are the resistivity of the barrier material (Ta/TaN) and copper, W is the line width, H is the copper thickness, and h_{Ta} is the barrier layer thickness.

4.3 Proposed EM-based aging sensor circuit

4.3.1 Wire structure for accurate EM-Induced aging detection

In the section, we investigate the new wire structures so that we can have more accurate detection of the EM induced resistance changes based on the new physics-based EM models. There are several factors we need to consider to design the right interconnect wire structures as the critical component of the aging sensors. First, there are the inherent variations in the metal wires, which will lead to the uncertainties in the nucleation time and the growth time. For a metal wire, its grain boundaries (GBs) may have different crystallographic orientations, which are characterized by different atomic diffusivities. Second, the grain sizes have a random distribution. As a result, the lifetime of the metal wires obeys the lognormal distributions [81]. Hence, we cannot use only one metal wire as the aging sensor. Third, how to design the geometry of the wires (length and width) to have a small area and power overhead for the sensors. We want the sensors to have a small footprint with considering their power consumptions and areas. In addition, they will meet the design rules, which are compatible with given design technologies. Fourth, we need to have an accurate estimation of the residual stresses σ_{Res} (mainly the thermal stresses), which largely depend on the temperature of the manufacturing process and even the packaging process.

In order to mitigate the inherent uncertainties in the atomic diffusivities in a metal line, one solution is to use a number of wires connected in parallel as shown in Fig. 4.2, in which each wire will have its own diffusion barriers at both ends (so they are treated as individual wires in the EM sense). However, they are connected in parallel by another

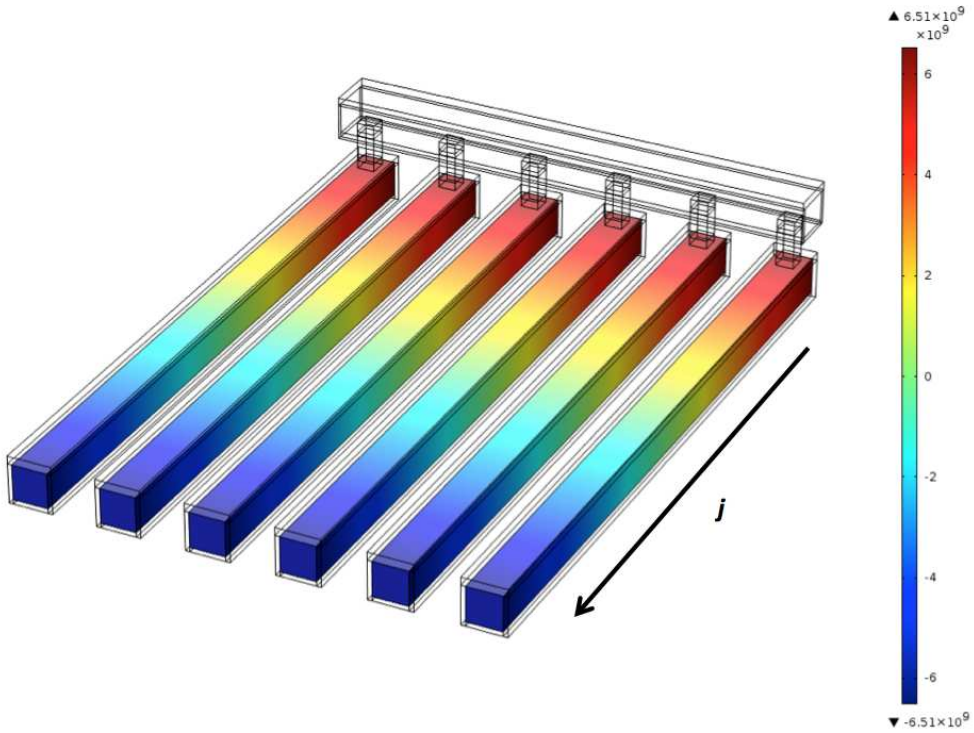


Figure 4.2: The proposed parallel multi-wire structure for the aging sensor and its stressed condition.

Note: the j represents the direction of electron flow

metal layer through vias. The color in the figure shows the simulated stress distributions in each wire. Those wires will be stressed all the time (with constant current running through them). We can design the wires such that they will fail at a specific time such as one year, two years, ..., or n years. Assume that the resistance value of the stressed wire at $t = 0$ is r_0. The failure time can be defined as the time when the wire resistance increases 10% of its values, i.e. $1.1 * r_0$, which can be predicted by the physics-based EM model. We also need to design a *reference* wire, which is not normally stressed (unless its resistance value is read during the detection time). The resistance value of the reference wire should be set to $1.1 * r_0/k$, k is the number of wires in the stressed wire set. For the reference wire, we

only need one wire segment as it will not age (the uncertainties in the EM-induced aging will not affect it). Our preliminary study shows that depending on the inherent variations, we can determine the number of wires such that we can confine the lifetime variations to a sufficient small range. The exact number wires used will be explored and validated by the actual silicon data. We remark that the intra-die process variations will affect the resistance values of those wires. However, if they are placed very closely, as this should be the case, the impacts will not be significant.

4.3.2 Resistance detection sensor circuit

Fig. 4.3 shows the schematic of the proposed EM-based aging sensor circuit. The circuit is composed of a constant current source, an EM stressed parallel wire (EMS) set, an EM reference wire (EMR), which will not be stressed in normal operation, a one-bit ADC (essentially an Opamp circuit), two resistors, one multiplex (MUX), one switch and one register to store the sensor digital output. The EMS contains a number of parallel wires (such as 6 in our initial analysis) with identical geometries. The EMR is just a single wire. The constant current source provides the current to stress EMS when the power is on. The one-bit ADC is used as a comparator to decide if the stressed EMS has a larger resistance than the EMR. If the voltage on EMS is higher than the voltage on EMR, it outputs 1, which indicates the failure happens, otherwise 0, which indicates that failure has not happened yet. The MUX and the switch are controlled by the *Read_en* signal from outside. When *Read_en* is off, the output of one-bit ADC will not be read into the register and the there is no current on EMR as it is in an open circuit. When Read_en is on, there will be a voltage on EMR, which is 10% higher than the original voltage on EMS and the

comparison result from one-bit ADC can be written into the register. As time going, the resistance of the EMS will increase due to the EM effect, which means the voltage of the non-inverting input of the one-bit ADC will increase. If the chip has been used for time longer than the designed failure time, the output of one-bit ADC will be 1 instead of 0.

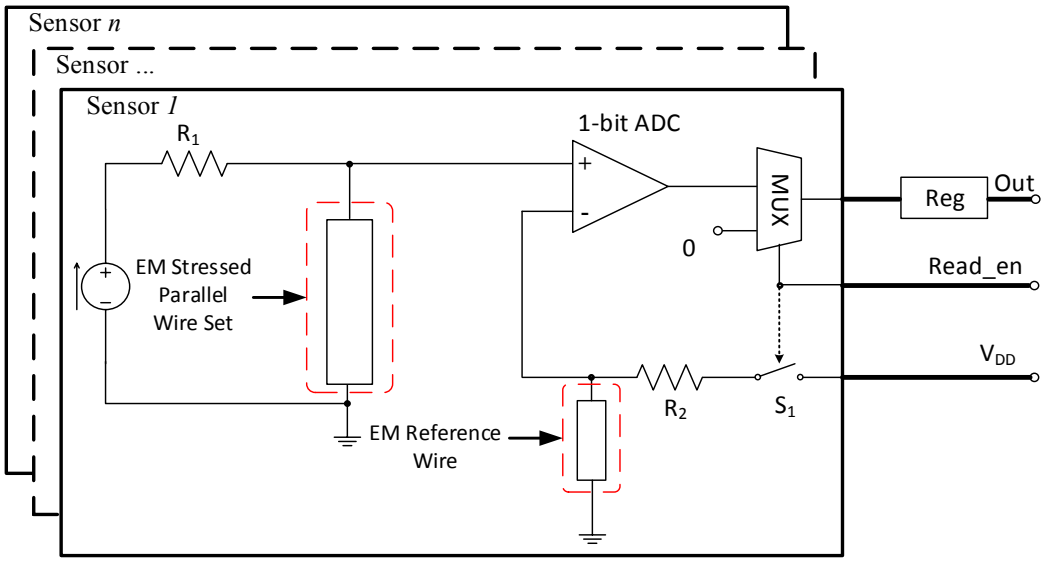


Figure 4.3: The structure of the EM-based aging sensor.

We will design a number of such aging sensors for the specific years (for instance 1 year to 10 years) in this project for validation purpose. Similar to the other on-chip aging sensors, the output registers can be connected by the JTAG circuits of the chip design so that one can read the aging information out in-situ during the testing or diagnosis time. The aging information can also be read out before the chips are put into the system. Note that the proposed EM-based sensor can automatically consider the temperature impacts on the lifetime of the chips as it is based on the EM aging effects.

4.4 Performance analysis and experimental results

In this section, we will present the performance analysis of the proposed EM-based aging sensor including simulation results.

4.4.1 Effect of number of wires

For the EM-based aging sensor, if the inherent variations are same for all the wires, then the wire, which fails at longer time, will have less absolute accuracy. For instance, 10% life time variation for a one-year wire will have an error of about one month, while 10% lifetime variation for 10-year wire will lead to errors about one year. In order to mitigate this problem, one solution is to add more wires for longer year wire set. Because more parallel wires we have, the smaller lifetime deviation the whole wire set will have. Fig. 4.4(a) and Fig. 4.4(b) show statistical analysis results for EM lifetime of stressed wire set connected in parallel versus number of wires in each set and different variations. These results come from 1000 Monte Carlo simulation runs and the aging sensor wires are set for the one-year lifetime. The EM-induced lifetime follows the lognormal distribution [81] and the variance are set to 0.001 and 0.002 respectively. With 0.001 variance and one wire, the EM lifetime will fall into $\pm 10\%$ lifetime mean with 99.83% chance and into $\pm 5\%$ lifetime mean with 88.64% chance. If we use 6 wires, there is 100% chance to achieve $\pm 10\%$ life mean and 98.66% chance for $\pm 5\%$ life mean, which is good enough. If we increase the variance to 0.002, then one wire can reach 97.46% chance for $\pm 10\%$ life time mean; 6 wires can achieve 99.95% chance for $\pm 10\%$ lifetime mean and 92.00% chance for $\pm 5\%$ lifetime mean. For 10 wires, we can achieve 99.99% chance for $\pm 10\%$ lifetime mean and 95.33% chance for

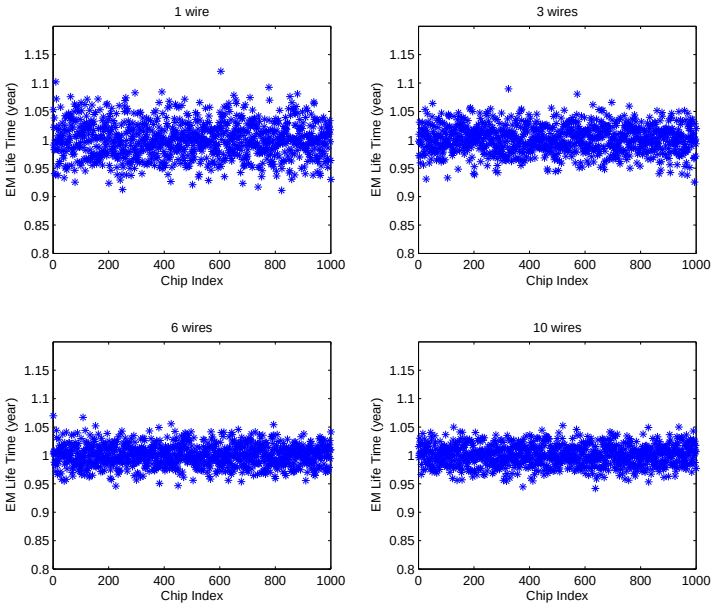
$\pm 5\%$ lifetime mean. As we can see, for large inherent variations, we have to increase the number of wires to mitigate lifetime variations. We remark that the intra-die environmental variations will also affect the resistance values of those wires. However, if they are placed very closely, as this should be the case, the impacts will not be significant.

Fig. 4.5 studies the lifetime variations versus the number of parallel wires used in each wire set for specific years (1 year, 6 years, 10 years). If we use the constant 6 wires for each set as shown in Fig. 4.5(a), we can see that the lifetime prediction variations in the 10-year wire set is quite significant for given variance ($\theta = 0.002$). But if we use varying numbers of wires for the same design (6 wires for 1 year, 10 wires for 6 years, and 14 wires for 10 years), the variations for the wire sets at the longer time lifetimes will be reduced as shown in Fig. 4.5(b).

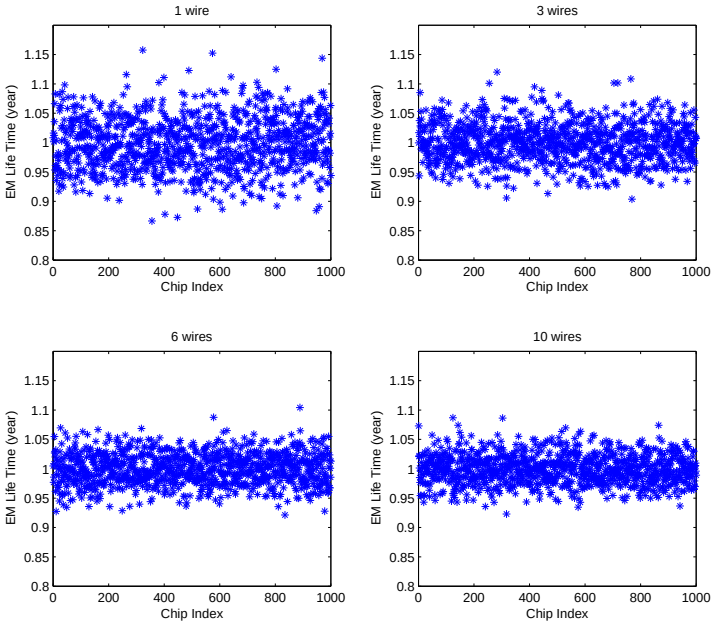
4.4.2 Effect of length of wires

Fig. 4.6 shows the relationship between wire length L and wire EM lifetime. The current density j is constant and set to $3 * 10^{10} A/m^2$. We show both the nucleation time and the growth phase time predicted by the new EM models. As we can see, the total lifetime increases with decreasing L (so does the area), which shows that shorter failure time will need larger area compared to the longer failure time.

For a specific failure year designed, the area for the sensor wires can be estimated as $A = W * L * k$, where W is the width of each stressed wire (assuming that all the stressed wire are same) and L is the length of the stressed wire. The area of reference wire

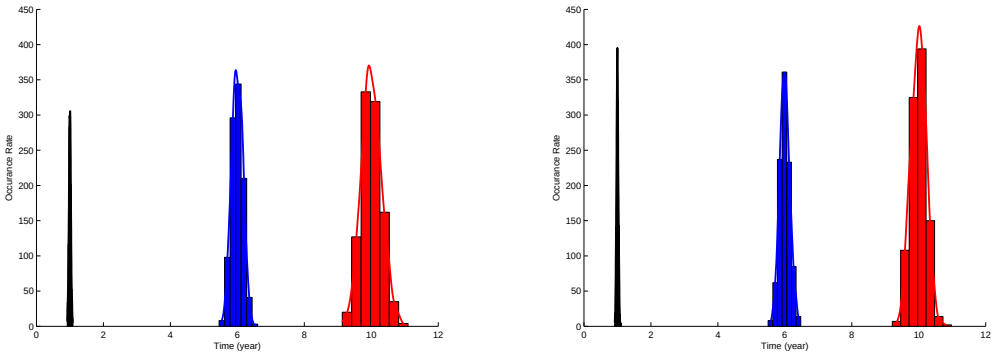


(a) Variance $\theta = 0.001$



(b) Variance $\theta = 0.002$

Figure 4.4: (a) and (b) the statistical study of stressed wires connected in parallel with different wire numbers and variations.



(a) Number of wires used: 6; Variance $\theta = 0.002$ (b) Number of wires used: varying; Variance $\theta = 0.002$

Figure 4.5: The statistical lifetime detections from the stressed wires: (a) using the constant 6 wires; (b) using the varying number wires (6 wires for 1 year, 10 wires for 6 years, and 14 wires for 10 years).

is $1.1 * W * L/k$, which is typically less significant compared to stressed wires. The power consumption for total stressed wires can be estimated as $P = k * I^2 * R{=}k * (j * A)^2 * \rho * L/A{=}k * j^2 * L * \rho * A{=}k * j^2 * L * \rho * W * H$, where ρ is the resistivity of the metal wire, j is current density, and H is the height of the wire segment as shown in Fig. 5.5. From the two formulas above, we can see that we should try to use the minimum width allowed by the technology node to save both area and power in theory. Our initial study shows that area and power are two performance metrics for trading-off in the design. Fig. 4.7 shows the power values versus the possible wire length (L) and current density j. The 10 red curves show the possible L and j values from 1 year to 10 years. We can clearly see the trade-off between L (area) and power.

Bear in mind that tens of EM-based aging sensor can be inserted into commercial chips, which would easily detect the counterfeit and recycled ICs and show the age of the

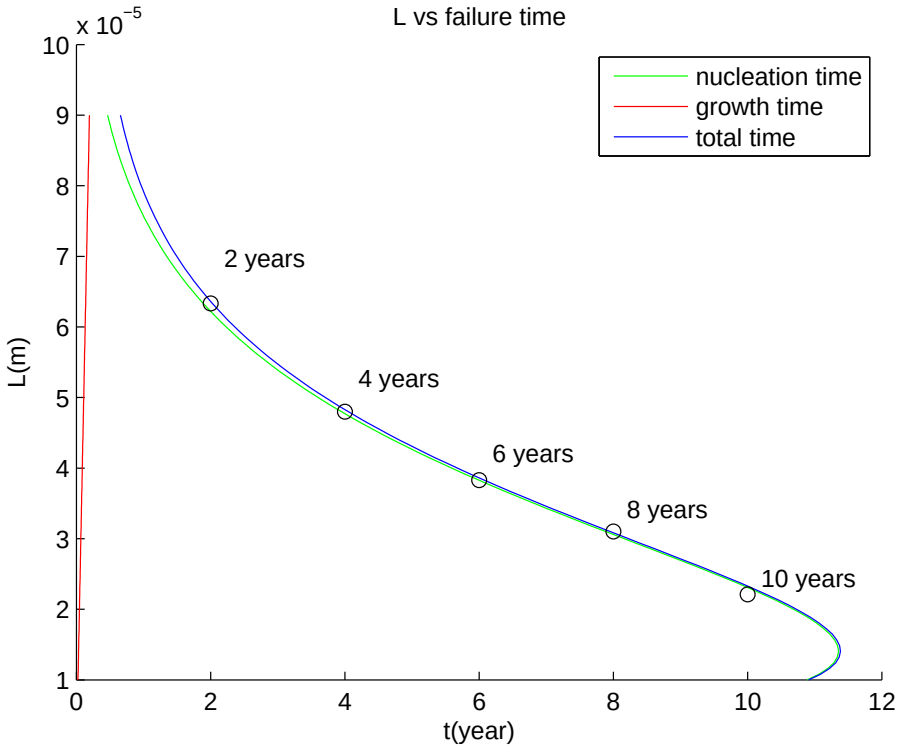


Figure 4.6: Length versus EM lifetime of a wire.

chip. Such a method is practical because the area overhead is small. An EM-based aging sensor with 10 stressed wires costs 100-500 um^2 with an SMIC 180nm technology, which depends on the length of the wire. Assuming a total of 10 sensors, the overhead is only 0.02% of the 25,000,000 um^2 area available in a 5 mm $\times$ 5 mm chip.

4.4.3 Experimental results

The proposed EM-based aging sensor circuit has been designed and validated using SPICE simulation. We performed 1000 Monte-Carlo simulation runs considering the variation of the nucleation time for the stressed wires. We design the wires such that

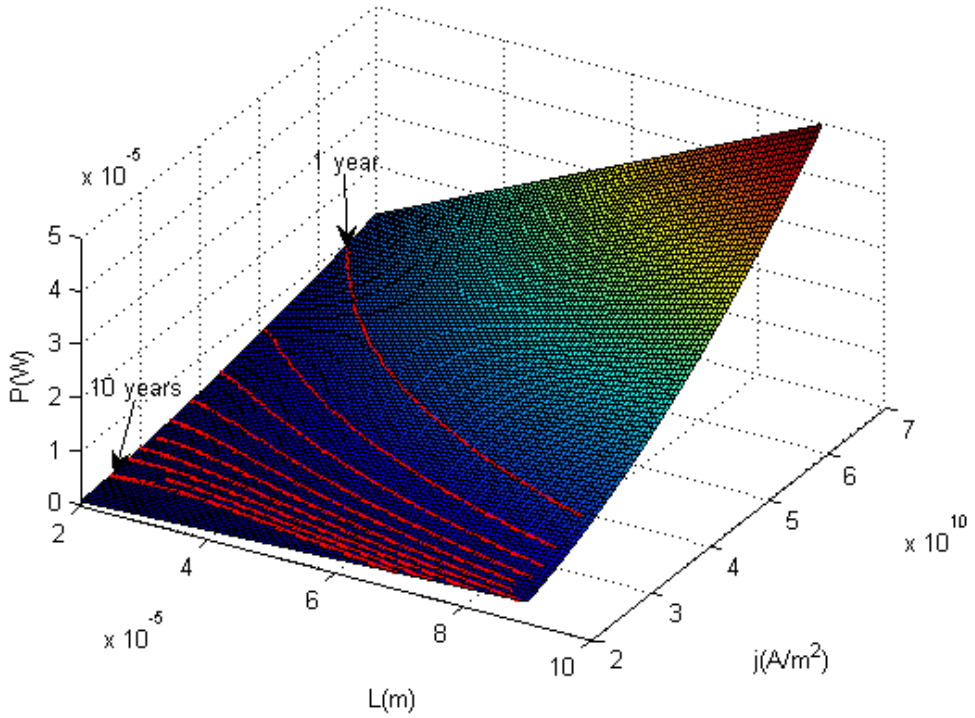
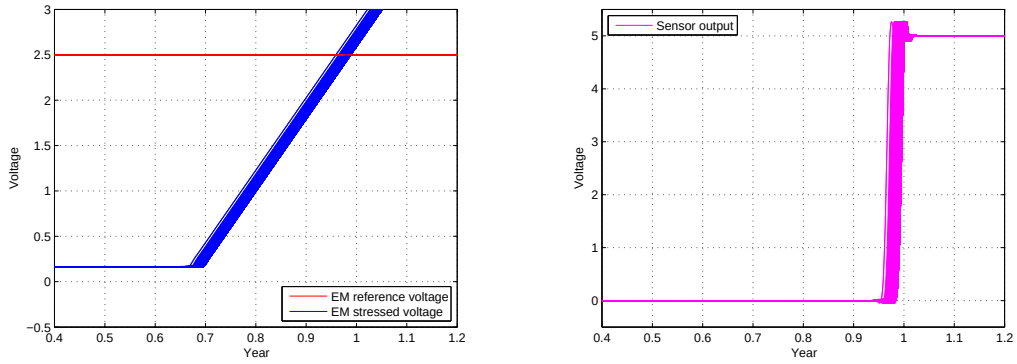


Figure 4.7: The power consumption of stress wires versus wire length and current density.

they will fail (its resistance increase just 10%) around one year with lognormal distribution (standard deviation is set to 0.001) in their nucleation times.

Fig. 4.8(a) shows the voltage waveforms at the two inputs of the ADC. In the beginning, the two inputs are clearly different. At the around 0.68 year, the voltages on the stress wires start to increase, which is also the nucleation time for the wires. Then, the voltage of the stressed wires starts to increase gradually until it runs across the 2.5 volts, and then the ADC output will change '1' from '0'. Fig. 4.8(b) shows that the ADC output will start to change from '0' (zero volts) to '1' (5 volts) around one year. As we can see, when the input voltage of stressed wires reaches the reference voltage, the output signal



(a) (b)

Figure 4.8: (a) The statistical voltage inputs for the ADC; (b) The statistical ADC output.

starts to change, which happens at one year in this case.

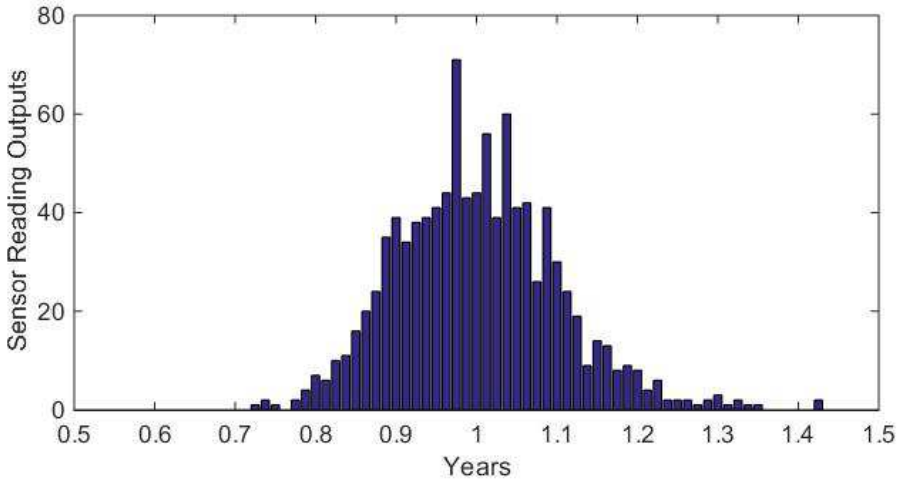


Figure 4.9: The statistical distribution of the lifetime of the sensor wires detected.

Fig. 4.9 shows the statistical distribution of the lifetime detection results of the sensor wires at the output of the ADC. As we can see, the distribution is lognormal owing to the lognormal distribution of the nucleation time of the wires. This clearly shows that the

proposed aging sensor work well as the failures of the wires can be very accurately detected around one year.

4.5 Summary

In this chapter, we have proposed a new on-chip aging sensor based on the EM-induced failure mechanisms to fast detect the recycled integrated circuits, which is one of the major hardware security issues facing the semiconductor industry. The new sensor is based on failure detections of DC current stressed metal wires to time the usage of chips over time. Compared with the existing ring-oscillator-based aging sensors, it can offer a simpler circuit implementation and smaller area footprints. It also provides a more accurate prediction of the chip usage time. The new aging sensor design is based on a newly proposed physics-based EM model. Experimental results show that the proposed aging sensor can accurately predict the targeted failure times in the presence of both inherent uncertainties. Our study also shows that more parallel wires will lead to more accurate statistical predictions at the cost of areas.

Chapter 5

Multi-Functional On-Chip Sensor for Comprehensive Detection of Counterfeit ICs

5.1 Introduction

The counterfeiting of integrated circuits (ICs) have become major problems in recent years, potentially impacting the security of electronic systems especially for military, aerospace, medical and other critical applications. Based on the 2008 report by the International Chamber of Commerce, the costs of the counterfeiting and piracy for G20 nations was as much as $775 billion per year and will grow to $1.7 trillion in 2015 [82]. The problem is getting worse due to deficiencies in the existing detection solutions and lack of effective avoidance mechanisms in place. Over the past couple of years, numerous reports [83] have

pointed to the counterfeiting issues in the U.S. electronics component supply chain. In addition to diminishing system dependability and usability, counterfeiting reduces the total revenue of companies from their research and development efforts, discourages innovation through the theft of intellectual properties (IPs), and produces low-quality products under established brand names [37].

Counterfeit ICs comes from different sources in the electronic supply chain. A counterfeit IC is not genuine one because it is an unauthorized copy; it does not conform to the original component manufacturer's (OCM) design, model, and/or performance; or it is not produced by the original component manufacturer or is produced by unauthorized contractors; it is an off-specification, defective, or used OCM product sold as "new" or working; it has incorrect or false markings and/or documentation [38, 84]. Therefore, counterfeit ICs can be classified into several major categories: (1) the recycled and remarked one, which is the most widely reported type of counterfeit parts; (2) the overproduced one, which is fabricated ones outside of contract by foundries; (3) the out of spec/defective one, which should be rejected during testing, but are stolen and sold on open markets; (4) The cloned one, which just copy the legal part by reverse engineering or illegal obtaining of IPs.

From detection techniques' perspective, the counterfeit ICs can also be categorized into defect ones and non-defect ones. The defect ones are those such as recycled/remarked ones and the out-of-spec/defective ones. Those counterfeit ICs will show some degrees of physical or electrical defects and anomalies due to aging and inherent defects from fabrications. Also the recycled ICs can cause reliability and security problems for many critical applications. Existing counterfeit detection techniques mainly focus on the detecting of the

defect ones the as the recycled and used ICs account for the majority of the counterfeit components [39].

On the other hand, the non-defect ones such as the overproduced, cloned ones are unauthorized productions of IC without the legal license and document. These kind of ICs may be exactly same as the authorized ones. The non-defect ones, however, undercut the competition with the unlicensed ones, which can cause significant revenue loss and related job loss for the original IC and IP owners and OCMs. Unfortunately, the existing detection techniques can only detect one type of those counterfeit ICs, not both. Therefore, it is urgently needed to develop new comprehensive, yet cost-effective, counterfeit IC detection techniques.

5.1.1 Review of existing detection method

For defect types, especially recycled and remarked ICs, there exists many detection techniques, which can be classified into physical methods and electrical methods [37]. Physical methods consist of incoming inspection methods such as visual inspection, X-ray imaging, package analysis method such as laser scanning microscopy, delid method, and the material analysis method such as using Fourier transform infrared, and X-ray fluorescence. Electrical methods contain the parameter tests, function tests, built-in tests and structural tests. In general, physical methods can be applied to all part types, but some of the methods are destructive and take hours to test. As a result, sampling is required to certify a batch of parts by observing a small number of parts. On the other hand, conventional electrical test methods are non-destructive and time efficient, yet they can be very expensive because such techniques are not necessarily designed for counterfeit detection.

One viable way for fast detect and effectively prevent the recycled chip is to insert a lightweight aging detecting sensor, which can directly tell the usage of the chips and some early efforts have been explored [42, 43, 44, 85].

Method in [43] designed the ring-oscillator(RO)-based aging sensor that relies on the aging effects of MOSFETs to change a RO frequency in comparison with the reference one embedded in the chip. As the chip ages owing to the wear-out mechanisms such as negative biased temperature instability (NBTI) and hot carrier injection (HCI), the shift threshold voltage of MOSFET devices, thus the frequency of RO indicates the level of aging, and provides a simple readout of the value. However, this method can only give very rough estimation of the usage age of the chip as the shift of the frequency depends on many factors. In order to mitigate this problem, the antifuse(AF)-based sensor was developed in [37]. The AF-based sensor essentially is a counter, which counts the clocks or derivatives of the clock events to log the usage of the chip. The antifuse memory is used to make sure the data in the count will not be erased or altered by attackers. However, the AF-based sensors suffer large area overhead especially when more accurate usage is required [37]. Another problem with this method is that it may not reflect the true aging-dependent usage of a chip. For instance, it will log the same usage time for a chip for different on-chip temperatures, however, which can have dramatically impacts on the aging effects from electromigration, NBTI and HCI [45]. Recently, on-chip aging sensor based on the electromigration (EM) failure mechanism of interconnect wires has been proposed [85]. The main advantage of EM-based aging sensor over RO-based aging sensor is that it can provide more accurate usage time estimation especially over longer period of time. The design is also simple and

light-weighted with small area and power overhead. But the EM-based sensor has more area overhead for estimation of short period time of usage as it needs longer interconnect wires.

For detection of non-defective counterfeit ICs, existing physical, electrical and aging sensor based methods will not be effective as there is no traceable properties can be detected. One potential solution is to have a post-fabrication authentication process in which, each IC will be uniquely registered into a global database using challenge-response pairs after fabrication and testing. The end users can verify the ICs for proper registration later. This post-fabrication authentication process is similar to the passive hardware metering method, which enables the design house to achieve post-fabrication control of the produced ICs [86, 87]. However, those methods cannot detect the recycled and used ICs.

5.1.2 New contribution

We develop an innovative multi-functional on-chip sensor and the related post-fabrication authentication methodology for detecting and preventing the counterfeit ICs. The proposed on-chip sensor can detect both recycled/remarked/out-of-spec chips, as well as cloned and over-produced ICs. It can serve as a central on-chip security hardware IP for counterfeit IC detection. In addition, it can be on-chip timer and post-fabrication authentication module and even activation module for ICs. Our new on-chip sensor has the following features:

- The new on-chip sensor combines an antifuse memory block, which is one-time programmable (OTP), with existing aging sensors. The memory block will not be used

as a counter as in the existing methods. Instead, it will store unique chip ID, time stamp of activation, and other important chip assets, which will be encrypted against tampering and can be verified by challenge-response pairs.

- Second, the new on-chip sensor combines the two types of aging sensors to detect both short-term and long-term aging effects so that it can be effective and area-efficient for both cases. The RO-based sensor is more effective for short-timer usage detection and EM-based aging sensor is more accurate for long term usage detection. The EM-based aging sensor exploits the natural aging/failure mechanism of interconnect wires to time the aging of the chip. It can serve as more accurate timer for the chip to meter the usage of long period time. As a result, it can enable timed service for some functionality of a chip and can also avoid the over-usage of authorized period for a chip or a system for certain security requirements. On the other hand, it can use only one of two sensors also based on the applications.
- Based on the new on-chip sensor, we propose a post-fabrication authentication methodology to detect and prevent the non-detective counterfeit ICs. All the fabricated ICs will be uniquely registered and activated with a unique chip ID in a global database. The unique chip ID will be written into the antifuse memory during registration process and chip will be activated after this process. In this way, it can prevent the cloned and over-produced ICs. In addition, it can get rid of the reference circuits in existing aging sensor designs as the initial electronic properties of the sensor circuits can be stored in the global database as well.

Simulation results shows the advantage of the proposed multi-purpose sensor against the existing on-chip sensors in terms of functionality, detection coverage, usage time estimation range and accuracy.

This chapter is organized as follows. In Section 5.2, we present the new multi-purpose on-chip sensor circuit architecture. Section 5.3 presents the overall authentication flow of the proposed on-chip sensor and detection methodology. Several statistical and variational analyses of the new sensors and comparison analysis are presented in Section 5.4. Last, Section 5.5 concludes.

5.2 The proposed on-chip sensor circuit

In this section, we present the architecture of proposed on-chip sensor circuit, which consists of one antifuse memory block, one aging sensor module, one encryption module and one activation module as shown in Fig.5.1. Each module will be discussed in detail in the following sections.

5.2.1 Antifuse memory block

An antifuse is an electrical device which performs the opposite function to a fuse. An antifuse starts with a high resistance and is designed to permanently create an electrically conductive path (typically when the voltage across the antifuse exceeds a certain level). It is an OTP memory technology. The antifuse memory is a type of read-only memory (ROM) meaning the data in them is permanent and cannot be changed. Antifuse is non-volatile, area and power efficient and has high reliability.

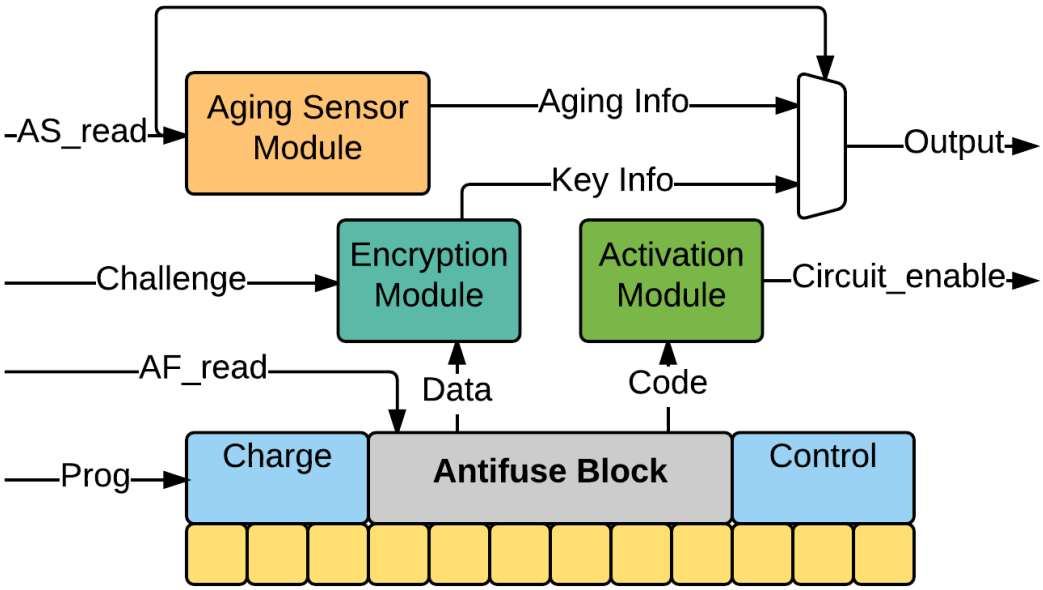


Figure 5.1: The architecture of proposed on-chip sensor

Most importantly, antifuse is confidential [88]. Before and after programming, the change on antifuse is extremely small, usually within tens of nanometers. In addition, there are millions of antifuse in one component. So the reverse engineering is almost impossible. As a result, it is ideal for storage the unique chip ID and activation time in the encrypted form. Additional, since the antifuse memory is on-chip, additional system design measures may be taken to make the device tamper-proof, such as password protecting the antifuse memory within the system chip. This newer memory technology provides unprecedented physical layer security. The antifuse memory has the following advantages:

- It's nonvolatile. Once we program the antifuse with a programming voltage, the antifuse is changed from one state to another. This process is irreversible. And the programmed antifuse state is permanent.

- It has radiation resistance [89]. Antifuse is a natural anti-radiation component. It can not only withstand the effects of nuclear radiation, but also be immune to the radiation from space particles.

- It has high reliability [90]. There is research indicating that antifuse FPGA can be an order of magnitude more reliable than the ASIC.

- It is confidential. Before and after programming, the change on antifuse is extremely small, usually within tens of nanometers. In addition, there are thousands, even millions of antifuse in one component. So the reverse engineering is almost impossible.

- It is area-efficient and power-efficient. With current technology, the antifuse can be very small. After programming, the resistance of the antifuse is only tens of Ohm. So the power consumption is also very small.

Credible evidence [88] of the CMOS antifuse's ability to hide information in silicon is illustrated in the three photographs in Fig 5.2. In the cross section view, the bit cell in the yellow rectangle is programmed. But it is not visible even under strong microscopy. The top view shows the deprocessing of XPM array. The diagonal bits in the yellow rectangle are programmed. But they are also invisible under microscopy. In the FIB voltage contrast, memory array in the yellow rectangle is programmed in a cheker-board pattern, but no bits are visible. Additional, since the antifuse memory is on-chip, additional system design measures may be taken to make the device tamper-proof, such as password protecting the antifuse memory within the system chip. This newer memory technology provides unprecedented physical layer security.

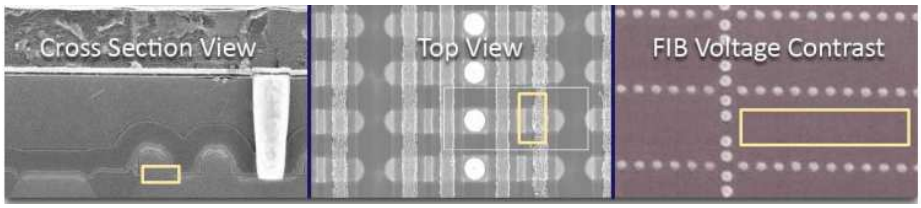


Figure 5.2: CMOS logic antifuse physical layer security

The antifuse memory block is used to store the unique key and other assets for each chip. The antifuse memories are programmed in a programming environment with relative high voltage. Therefore, integrated charge pumps are used to provide sufficiently high voltage in embedded antifuse memories. We use existing antifuse blocks instead of designing a new one.

5.2.2 Aging sensor module

Two different aging sensors to identify recycled ICs are used in this aging sensor module. The RO-based sensor is based on the aging effects on RO. The usage time can be detected by degraded RO frequency. The EM-based sensor relies on the EM aging effects on interconnect wires. The resistance change of the stressed wires can be used to estimate the chip usage time. The RO-based aging sensor is used to detect short-term aging while the EM-based aging sensor is used to detect long-term aging. In addition, the EM-based aging sensor can serve as a timer which can be used to disable the chip after a certain time. The two aging sensors will be discussed in detail in the following sections.

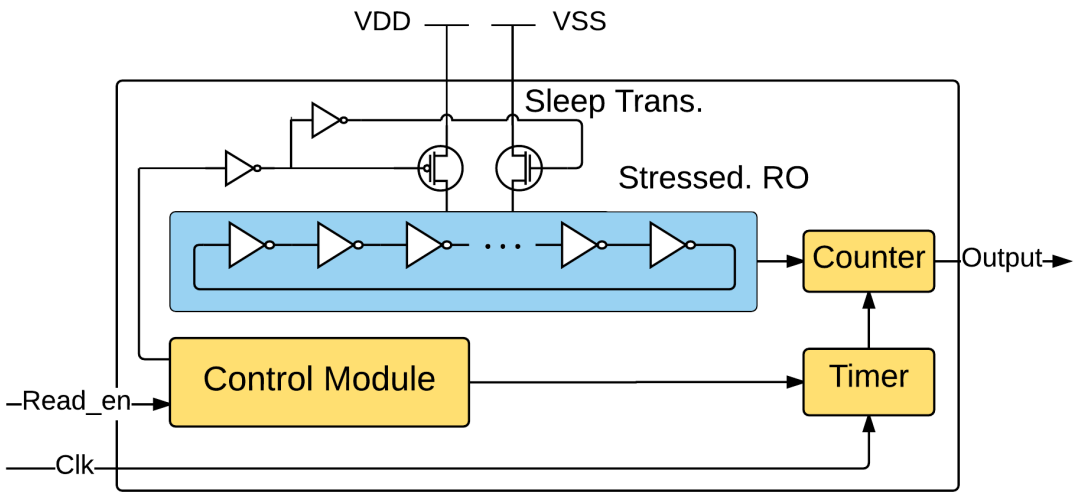


Figure 5.3: Structure of RO aging sensor

RO-based aging sensor

In the new sensor design, the new RO-based sensor shown in Fig. 5.3 follows the similar design in [43]. But different than the one in [43], the new RO-based sensor only has one RO (compared to the two in the existing works [43], as the reference frequency will be stored in the design house database and can be assessed when the chip ID is read back by challenge-response pairs during the authentication process. The details of the whole flow will be discussed in the following section. Fig. 5.4 shows the typical frequency change over time for the RO-based sensor. As we can see, as time goes by, the frequency change rate goes down, which means that the sensitive of frequency change become smaller and it will more difficult and less accurate to estimate usage time based on the frequency changes for long period of time.

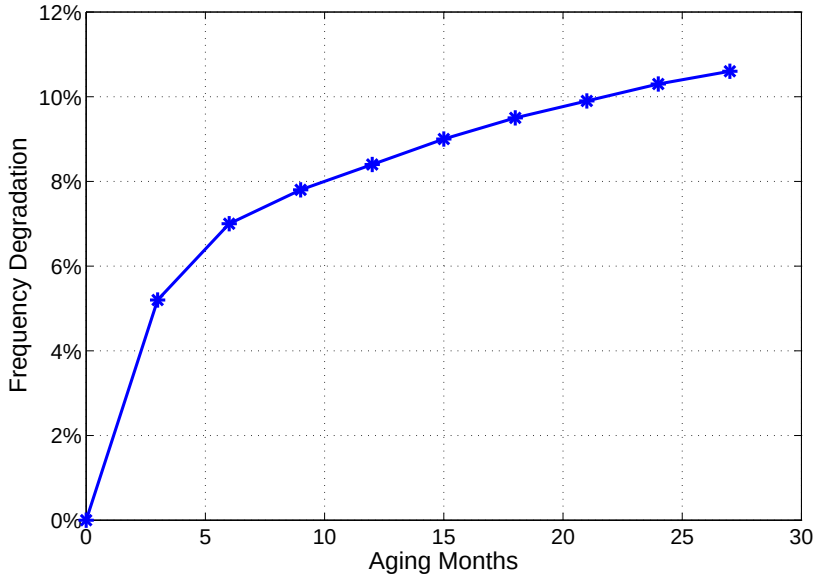


Figure 5.4: Frequency degradation of a 5-stages RO

EM-based aging sensor

Fig. 5.8 shows the schematic of the proposed EM-based sensor. The design follows the recent work in [85]. Fig. 5.5 is the multi-wire structure to be stressed for a specific time in one sensor. The EM-based sensor has two versions. One version is aging sensor shown Fig. 5.6. In this case, we have a group of wires connected in parallel and stressed by DC current. The current densities in the wires are setup so that the wires will be nucleated at a specific time (e.g., one year or 10 years). The initial resistance of the wires will be stored in the design house database as a references. When the resistance of wires changes by 10%, it can be counted as the failure and time difference between the activation time and current is the usage time. Another version of the EM-based sensor is the timer version as shown

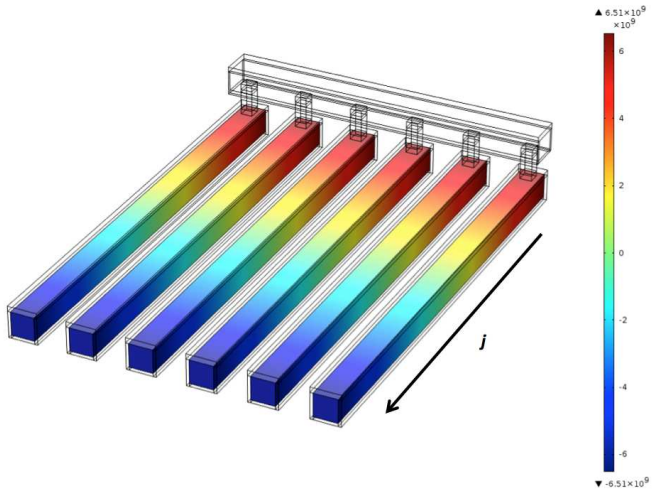


Figure 5.5: The multi-wire structure

Fig 5.7. In this case, the reference wire, which has the same geometry as the stressed wires is used. The sensor can output the signal when the wire resistance change significantly (by 10%). The signal can be used to lock the chip or lock certain functions of the chip for timed-service of the chip or the system.

5.2.3 Encryption and Activation module

The encryption module is used to encrypt the data from antifuse blocks with the challenge from the design house database. This module can be any existing encryption module, e.g., AES method. It is used to make sure the key information in the antifuse block cannot be directly accessed by any adversary.

The proposed on-chip sensor also allows one-time activation of a chip or certain chip functions. This is achieved by the activation module. Once the chip pass the post-fabrication testing, design house can write the key into the anti-fuse blocks. There are many ways to implement the chip-level activation process [91, 92]. For instance, we check the

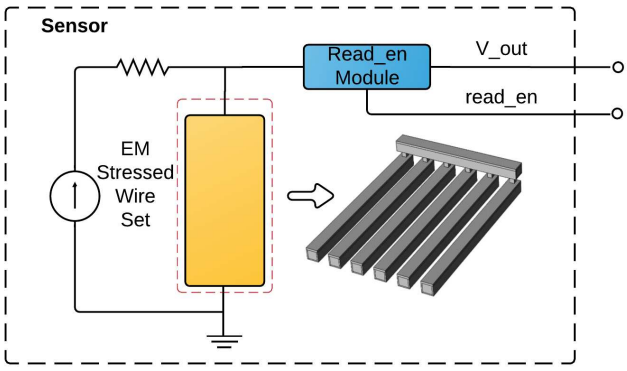


Figure 5.6: The EM sensor-only circuit

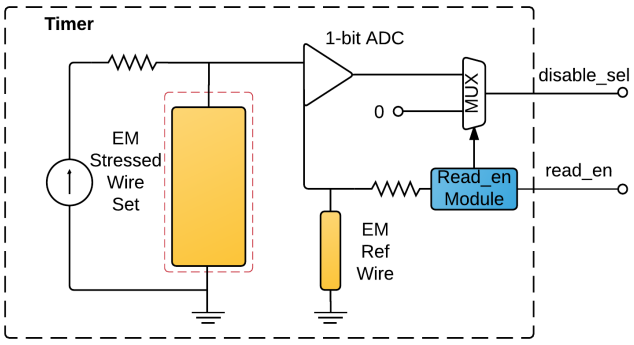


Figure 5.7: The whole aging sensor with multiple EM sensors and timers

parity of the bits of the stored key in anti-fuse memory. We can also check number of zeros or number of ones as well (bit stream written into antifuse memory needs to enforce some properties in this case). The checking circuit inside the activation module can be obfuscated for further protection. The output of the activation module can drive a randomly scatted XOR gates in a chip to enable the unlocking process. In this way, it is difficult to modify layout by counterfeiters.

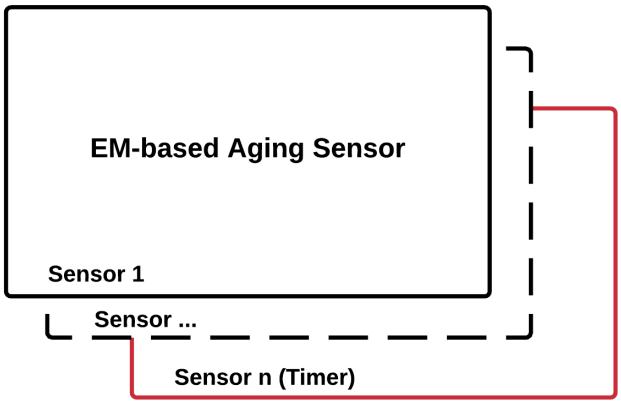


Figure 5.8: The structure of EM aging sensor

5.3 The proposed counterfeit IC detection methodology

In this section, we present the proposed overall counterfeit IC detection methodology and the IC authentication flow based on our on-chip sensor with antifuse memory.

Fig.5.9 shows a typical lifetime process that an IC goes through, which include the design, fabrication, assembly, distribution, usage in the system until end of its life. As one can see, there are vulnerabilities associated with each step in this supply chain. In the design stage, an IP can be stolen and cloned. In the fabrication process, an IC can be overproduced. In the assembly phase, out-of-spec/defective ICs can be sold to open market by an untrusted assembly. Illegal activities during distribution, in-the-system (lifetime) may bring different types of counterfeits back into the supply chain (recycled, remarked, etc.).

To detect the all non-defective counterfeit ICs, we propose a new supply chain flow with post-fabrication authentication process as shown in Fig. 5.10. Basically, one

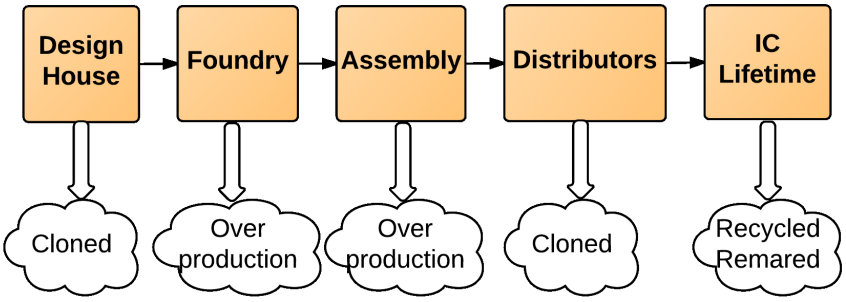


Figure 5.9: The electronic component supply chain and vulnerabilities

needs to break the flow from assembly to the distribution. As shown in Fig. 5.11, once chips has been tested and packaged in the assembly stage, they will be sent back to the design house. After a functional test, for the non-defective ICs, a unique chip ID, activation time and other assets will be written into the antifuse memory in the on-chip sensor. And the initial aging reference properties will be stored into the design house global database for future verification. All the information cannot be directly accessed and will be encrypted using standard cryptographic method to prevent the attack and tampering. Also during this process, the design house can activate the locked chip, which will not work after the fabrication process, using the unique content in the antifuse memory. In this way, the design house can have better control of the ICs to prevent cloning and other unauthorized use.

Fig. 5.12 shows the proposed comprehensive detection policy for counterfeit ICs. In general, a newly fabricated chip needs to pass two tests to be proved as a fresh and authentic IC. The first test is called *fingerprint* test. The design house device database generates a random challenge which can be input into the IC. If the IC cannot generate any

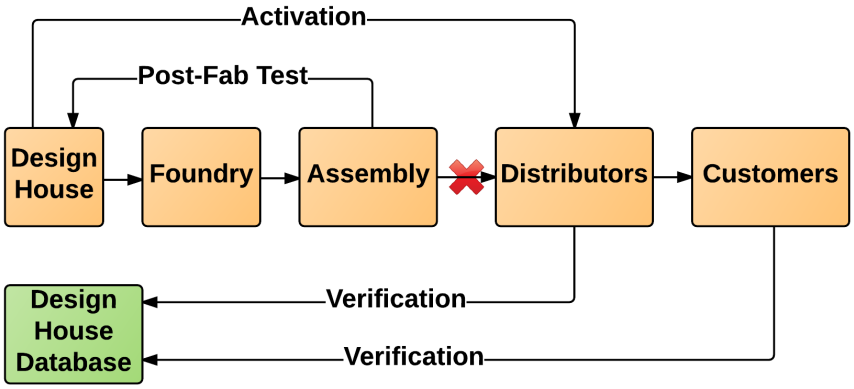


Figure 5.10: The proposed supply chain with post-fabrication authentication

response or just output a incorrect response, then it haven't taken the official design house antifuse activation. It means this IC never comes back to the design house after fabrication. So it can be detected as overproduced or cloned IC. If the response of the IC matches the information in the design house database, then we can get its production information. By comparing the antifuse production information and the device footprint information, it's easy to detect it's a remarked IC or not. The second test is called aging test. This test is performed to detect recycled or used IC or tell the user the estimated usage time of the chip. By reading the aging sensor output, we can detect if it's recycled IC or not. Based on the aging model of the aging sensors employed and aging output, we can determine time usage of the chip accurately.

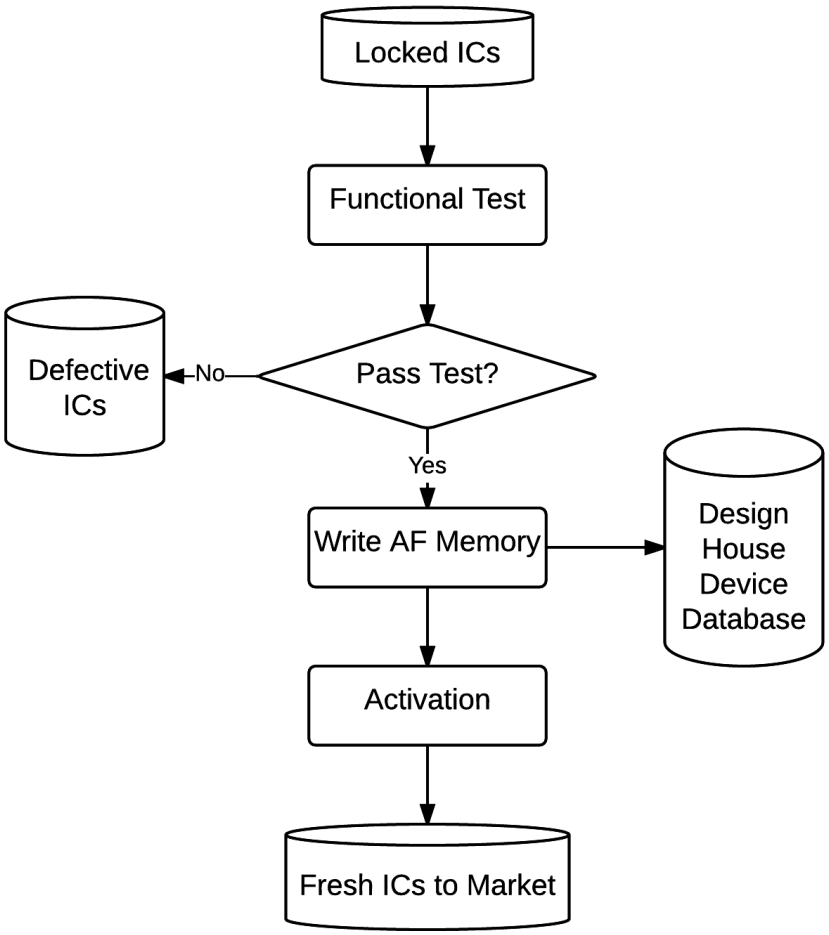


Figure 5.11: The proposed post-fabrication authentication

5.4 Numerical results and discussions

In this section, we first summarize the feature comparison among different sensors, then we will present the simulated results of the RO-based and EM-based sensors. The performance and overhead analysis will be discussed.

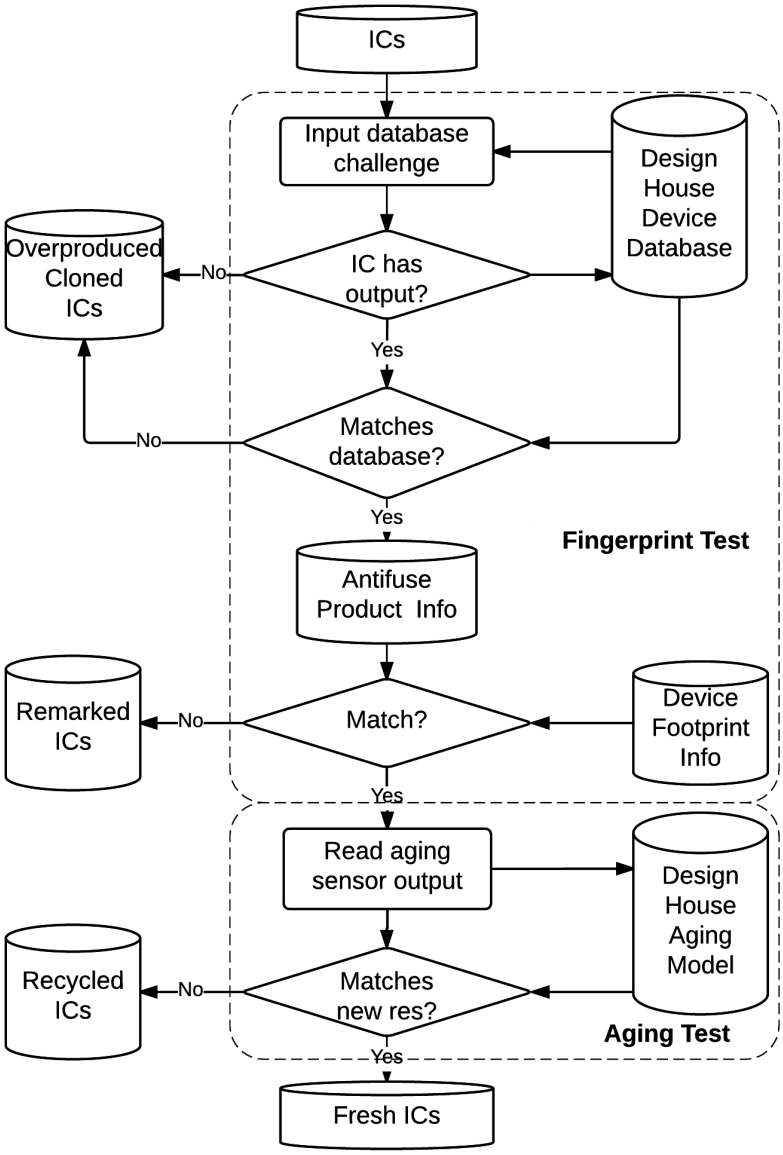


Figure 5.12: The proposed comprehensive detection methodology for counterfeit ICs

The feature and function comparison among different sensors

Table 5.1 summarizes the major feature comparison among the RO-based, EM-based and the proposed hybrid aging sensor. The RO-based sensor has high short-term

Table 5.1: Aging Sensor Comparison

Feature	RO	EM	Proposed
Short-term usage accuracy	high	low	high
Long-term usage accuracy	low	high	high
Post-fabrication auth	no	no	yes
Detect cloned and over-produced ICs	no	no	yes
Reference circuit	needed	needed	not needed
Activation	no	no	yes
Timed-serviec	no	no	yes

usage accuracy but low long-term usage accuracy. The EM-based sensor has high long-term usage accuracy if we use multiple stressed wires. However, its design is not good for short-term recycled IC detection. Our proposed hybrid aging sensor can maintain high accuracy for both short- and long-term recycled IC detection. The proposed sensor can also allow post-fabrication authentication to detect cloned and over-produced ICs. It also allows activation of the chip and timed services for ICs when it is used as on-chip timer.

5.4.1 Results for RO-based aging sensor

The RO-based aging sensor has been implemented and simulated using HSPICE MOSRA from Synopsys. In our implementation, we selected 7-stage and 15-stage ROs to compare the results. In order to model the variation, we performed Monte Carlo(MC) simulation with 1,000 samples of the RO in HSPICE.

Similar to the simulation in [43], we considered two process variations to investigate the impact of variation on the detection of the recycled ICs. Table 5.2 shows the different process variations used in our simulation. RO-based sensors with 7/15-stage ROs are simulated at 25°C with PV0/1. PV0 represents the expected process variation between ROs while PV1 is the worst-case scenario. Thousand sensors are generated using MC simulation by HSPICE and the total aging time is set at 15 months with a 3-month step.

Table 5.2: Process variations used

	Inter-die			Intra-die		
	Vth(%)	L(%)	Tox(%)	Vth(%)	L(%)	Tox(%)
PV0	5	5	2	5	5	1
PV1	20	20	6	10	10	3

Fig. 5.13 shows the simulation results for the RO-based aging sensor. The x-axis represents the frequency difference ($f_{diff} = f_{init} - f_{stressed}$) between the initial value and the stressed RO. Note that we don't need reference RO because we store the initial frequency in the global database. The y-axis represents the frequency of occurrence. The legend in the figures denotes the aging time (for example, AT = 3M denotes the RO is aged for 3 months). The green distribution represents the f_{diff} distribution for the new ICs where the RO has not been aged and is centered at 0 MHz. The light blue and dark blue distributions represent 3 months and 15 months of aging respectively. It is clear that aging shifts the distributions to the right as the stressed RO has aged more and become slower resulting in the right shift of f_{diff} distribution.

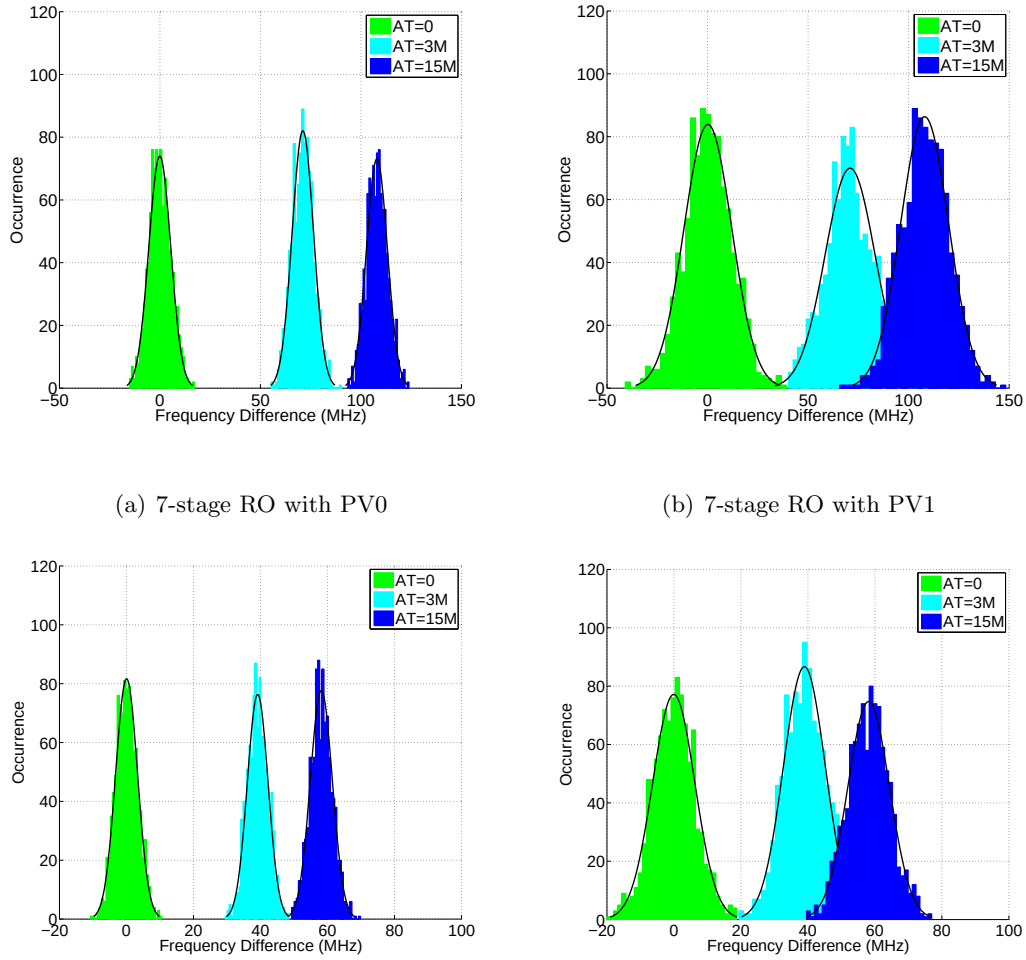
We can clearly identify recycled ICs when the two distributions ($T = 0$ and $T = 3,15M$) do not overlap with one another. In Fig. 5.13(a), after being used for 3 months, the stressed RO suffers from aging effects and its frequency became lower. The lowest frequency difference between the new and the stressed ROs is larger than the largest frequency difference present in the new IC set. Therefore, the recycled IC detection rate for ICs aged for 3 months or longer is 100%. At 15 months, the frequency differences between the new and the stressed ROs can be larger.

Fig. 5.13(b) shows the frequency difference occurrence rate between the 7-stage new and stressed ROs with process variations PV1. Moving from PV0 to PV1, interdie and intradie variations both become larger. As process variation increases, the variance in f_{diff} grows, which results in an overlap between 0 and 3,15M distributions. In this case, we should expect higher mis-prediction rates.

The simulation results for 15-stage ROs using same process variations are shown in Fig. 5.13(c) and Fig. 5.13(d). Comparing to the 7-stage ROs, the frequency difference between aged and new ICs is smaller because we use the larger stage ROs. Although it impacts the absolute value of the frequency difference, the detection rate will not be impacted significantly.

5.4.2 Results for EM-based aging sensor

The EM-based aging sensor circuit is designed and validated using SPICE simulation. We performed 1000 Monte Carlo simulation considering the variation of the failure time for the stressed wires. The failure time can be defined as the time when the wire



(a) 7-stage RO with PV0

(b) 7-stage RO with PV1

(c) 15-stage RO with PV0

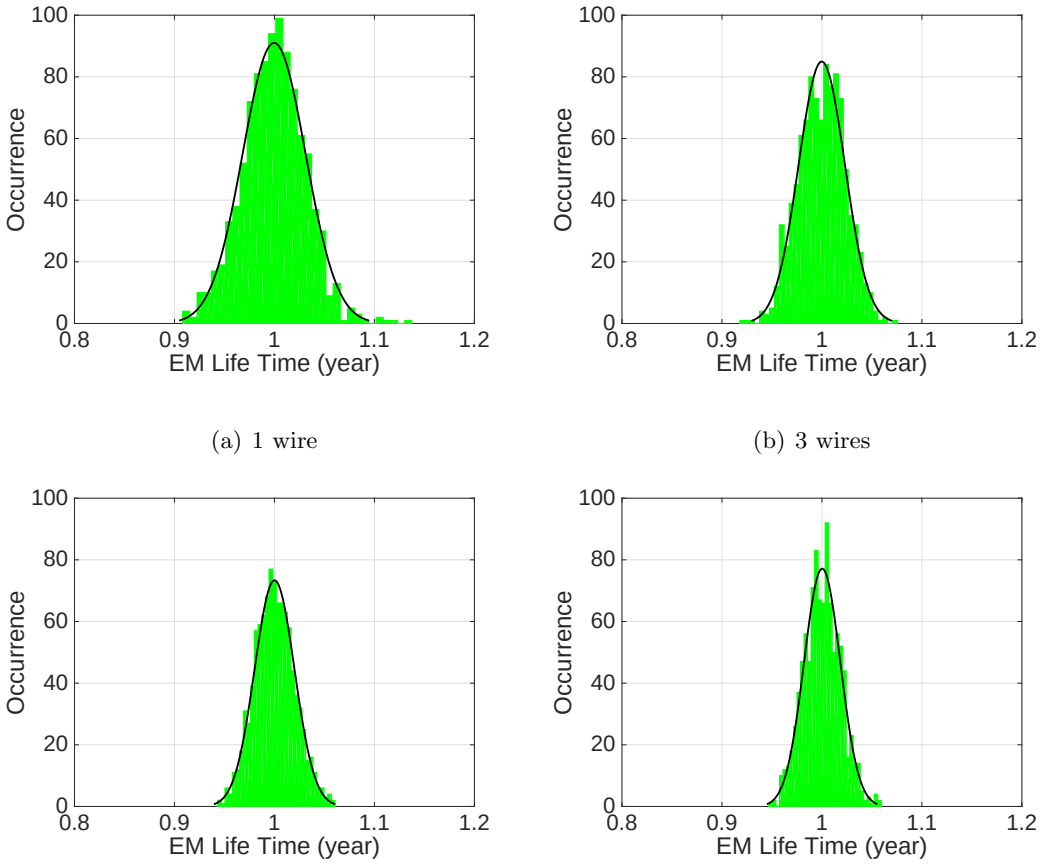
(d) 15-stage RO with PV1

Figure 5.13: Process variation impacts on frequency spreading and recycled IC detection probability.

resistance increases 10%, which can be predicted by the physical-based EM-model [78, 46].

To verify the effects of aging on an EM-based aging sensor, we performed 1,000 MC simulation. The simulation is conducted using HSPICE and MATLAB with the physical-based EM-model. The EM stressed wire sets are composed of 1, 3, 6 and 10 wires which will fail around one year. The EM failure time follows lognormal distribution [81].

The 1000 MC simulation results of the EM-based aging sensor are shown in Fig. 5.14. The variance of the lognormal distribution is set to 0.001. With 0.001 variance, we can see that with one wire, the EM lifetime will fall into ±10% lifetime mean with 99.83% chance and into ±5% lifetime mean with 88.64% chance. If we use 6 wires, we can have 100% chance to achieve ±10% life mean and 98.66% chance for ±5% life mean, which is good enough. As we can see, we can mitigate the failure time variations by increase the number of wires.



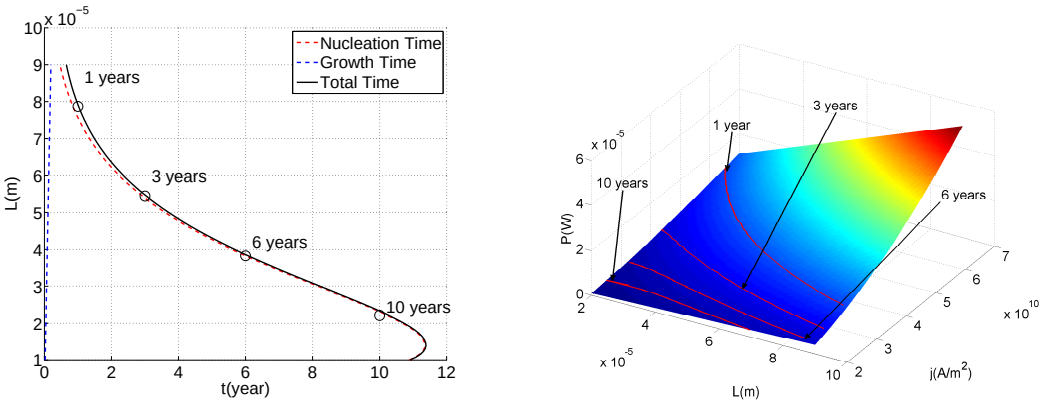
(a) 1 wire

(b) 3 wires

(c) 6 wire

(d) 10 wires

Figure 5.14: The statistical study of stressed wire set with different wire numbers



(a) (b)

Figure 5.15: (a) Length versus EM lifetime of a wire. (b) The power consumption of stress wires versus wire length and current density.

Fig. 5.15(a) shows the relationship between wire length L and wire EM lifetime. The current density j is constant and set to $3 * 10^{10} A/m^2$. We show both the nucleation time and the growth phase time predicted by the physics-based EM models. As we can see, the total lifetime increases with decreasing L (so does the area), which shows that shorter failure time will need larger area compared to the longer failure time.

Fig. 5.15(b) shows the power values versus the possible wire length (L) and current density j. The 4 red curves show the possible L and j values for 1 year, 3 years, 6 years and 10 years. We can clearly see the trade-off between L (area) and power.

5.4.3 Performance analysis and comparison

Fig. 5.16 shows the typical frequency change over a long period of time for a 5-stage RO-based sensor. As we can see, the frequency change rate is very high at the beginning, which is helpful to detect recycled ICs for a short period of time. However, as time goes by, the frequency change rate goes down, which means that it will be more difficult and less

accurate to estimate usage time for long period of time. Considering the process variances, 1% frequency difference can lead to a big estimated usage time region (30 months as shown in Fig. 5.16). So the RO-based sensor is not a good timer for long period use. In contrast,

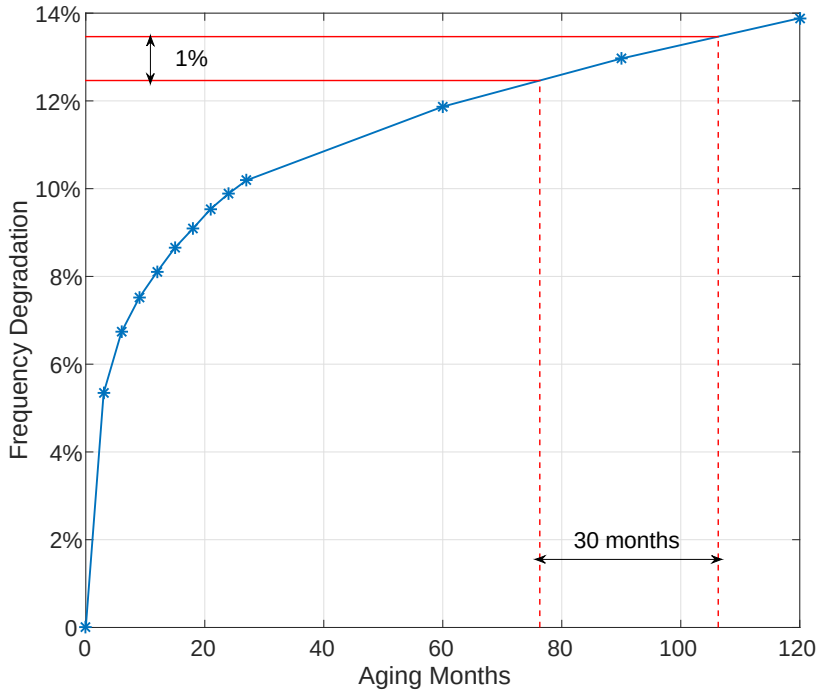


Figure 5.16: The RO-based aging sensor error rate for long period time

the EM-based sensor can be a good long-period time because of its accuracy. The estimated usage time region for a long period can be very small if we use multiple stressed wires [85]. In addition, in Fig. 5.15(b), comparing to the 1-year EM-based sensor, the 10-years EM-based sensor has smaller L(area). So the EM-based sensor for long-term use is also area-efficient.

Bear in mind the hybrid aging sensor can be inserted into commercial chips, which would easily detect the recycled ICs and show the age of the chip. Such a method is practical

because the area overhead is small. The RO-based sensor only takes n invertors, where n is the number of stages in RO. Its area is equal to tens of NAND2 gates, which is negligible to the whole chip. An EM-based aging sensor with 10 stressed wires costs 100-500um^2 with an SMIC 180nm technology, which depends on the length of the wire. Assuming a total of 5 EM-based sensors, the overhead is only 0.01% of the 25,000,000 um^2 area available in a 5 mm $\times$ 5 mm chip.

5.5 Summary

In this chapter, we have propose a multi-functional on-chip sensor and the corresponding methodology for detecting and authenticating the counterfeit ICs. The proposed on-chip sensor can detect both recycled/remarked/out-of-spec chips, as well as cloned and over-produced ICs. The new on-chip sensor, which combines aging sensors with antifuse memory, can serve as a central on-chip security hardware IP for counterfeit IC detection, on-chip timer and post-fabrication authentication and even activation module for ICs. On top of the new sensor hardware, we propose a post-fabrication authentication process to detect and prevent the non-detective counterfeit ICs. All the fabricated ICs will be uniquely registered and activated with a unique chip ID in a global database. The unique chip ID will be written into the anti-fused memory during registration process and chip will be activated after this process. Simulated results show the advantage of the proposed multi-purpose sensor against the existing on-chip sensors in terms of functionality, detection coverage and usage time estimation range and accuracy.

Chapter 6

Conclusion

The linear algebra is an important part in circuit simulation. In spite of many research works for parallel dense linear algebra support, the sparse linear algebra support is still limited. GPU-accelerated sparse linear algebra as important technique for circuit simulation is still far away from a wide industrial acceptance.

On the other hand, the counterfeit ICs have become a major security threat for commercial and mission-critical systems. Counterfeit ICs comes from different sources in the electronic supply chain and can be categorized into many types. Although there exist many detection and prevention methods for each type of counterfeit IC, there's no general solution for all of them.

This thesis proposes several important parallel sparse linear algebra operations, such as sparse matrix-vector multiplication, direct LU factorization and iterative general minimum residual linear solver. The proposed algorithms and implementations have significant speed-up over existing solutions. This thesis also establishes a counterfeit IC detection

methodology. The corresponding multi-functional on-chip sensor and post authentication policy are able to detect all counterfeit IC types with good accuracy. In this chapter, the main contributions of the thesis are summarized.

6.1 Summary of Research Contributions

6.1.1 GPU-accelerated sparse linear algebra for VLSI systems

In Chapter 2, we propose a new sparse LU solver on GPUs for circuit simulation and more general scientific computing. The new algorithm is based on a hybrid right-looking LU factorization method, which we show, is more suitable for GPU computing as it can exploit more parallelism than the widely used left-looking LU factorization algorithm. We further showed how the three loops of LU factorization can be parallelized based on the GPU thread-block structures, while the existing GPU left-looking LU factorization method can only parallelize two loops. Numerical results show that the proposed GLU solver can deliver 5.71× and 1.46× speedup over the single-threaded and the 16-threaded PARDISO solvers respectively, 19.56× speedup over the KLU solver, 47.13× over the UMFPACK solver and 1.47× speedup over a recently proposed GPU-based left-looking LU solver on the set of typical circuit matrices from University of Florida Sparse Matrix Collection (UFL). Furthermore, we also compare the proposed GLU solver on a set of general matrices from UFL, GLU achieves 6.38× and 1.12× speedup over the single-threaded and the 16-threaded PARDISO solvers respectively, 39.39× speedup over the KLU solver, 24.04× over the UMFPACK solver and 2.35× speedup over the same GPU-based left-looking LU solver. At last, comparison on self-generated RLC mesh networks shows a similar trend,

which further validates the advantage of the proposed method over the existing sparse LU solvers.

In Chapter 3, we propose an efficient parallel dynamic linear solver *GPU-GMRES*. The new solver is based on the preconditioned GMRES solver implemented CPU-GPU platforms. The proposed GPU-GMRES solver is based on the very general and robust Incomplete LU based preconditioner. We have shown that by properly selecting the right amount of fill-ins in the LU factors, a good trade-off between GPU efficiency and convergence rate can be achieved for the overall best performance. In addition, a new fast parallel SpMV multiplication algorithm is proposed to further accelerate the GMRES solver. The new algorithm, called *segSpMV*, can enjoy full coalesced memory access. To further improve the scalability and efficiency, *segSpMV* method is further extended to multi-GPU platforms. The resulting *multi-GPU segSpMV* can deliver further performance enhancement for the resulting *multi-GPU-GMRES* solver. Furthermore, we have properly partitioned the major computing tasks in GMRES solver to minimize the data traffic between CPU and GPU, which further boosts performance of the proposed method. Experimental results on the set of the published IBM benchmark circuits and mesh-structured power grid networks have shown that the GPU-GMRES solver can deliver order of magnitudes speedup over one direct LU solver. The resulting *multi-GPU-GMRES* can also deliver 3-10x speedup over the CPU implementation of the same GMRES method on transient analysis.

6.1.2 Potential solutions to mitigate the counterfeit IC problem

In Chapter 4, we have proposed a new on-chip aging sensor based on the EM-induced failure mechanisms to fast detect the recycled integrated circuits, which is one of the

major hardware security issues facing the semiconductor industry. The new sensor is based on failure detections of DC current stressed metal wires to time the usage of chips over time. Compared with the existing ring-oscillator-based aging sensors, it can offer a simpler circuit implementation and smaller area footprints. It also provides a more accurate prediction of the chip usage time. The new aging sensor design is based on a newly proposed physics-based EM model. Experimental results show that the proposed aging sensor can accurately predict the targeted failure times in the presence of both inherent uncertainties. Our study also shows that more parallel wires will lead to more accurate statistical predictions at the cost of areas.

In Chapter 5, we have propose a multi-functional on-chip sensor and the corresponding methodology for detecting and authenticating the counterfeit ICs. The proposed on-chip sensor can detect both recycled/remarked/out-of-spec chips, as well as cloned and over-produced ICs. The new on-chip sensor, which combines aging sensors with antifuse memory, can serve as a central on-chip security hardware IP for counterfeit IC detection, on-chip timer and post-fabrication authentication and even activation module for ICs. First, the new sensor adds an antifuse memory into existing aging sensors to reduce reference circuit related area overhead of those sensor circuits as initial electronic properties of sensor circuits are stored in a global database, accessed by unique chip via challenge-response pairs. Second, the new sensor can combine both the RO-based sensor with recently proposed EM-based aging sensor so that it can be effective for chip usage estimation for both short and long period. As a result, it can serve as more accurate timer for the chip to meter the usage of long period time, which can allow for timed services of some functionality of a chip in

addition to detection of the recycled/remark ICs. On top of the new sensor hardware, we propose a post-fabrication authentication process to detect and prevent the non-detective counterfeit ICs. All the fabricated ICs will be uniquely registered and activated with a unique chip ID in a global database. The unique chip ID will be written into the anti-fused memory during registration process and chip will be activated after this process. Simulated results show the advantage of the proposed multi-purpose sensor against the existing on-chip sensors in terms of functionality, detection coverage and usage time estimation range and accuracy.

Bibliography

[1] A. Kteyan, V. Sukharev, M. A. Meyer, E. Zschech, and W. D. Nix, "Microstructure Effect on EM-Induced Degradations in Dual-Inlaid Copper Interconnects," *Proc. AIP Conference*, vol. 945, pp. 42–55, 2007.

[2] V. Sukharev, "Physically Based Simulation of Electromigration-Induced Degradation Mechanisms in Dual-Inlaid Copper Interconnects," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 1326–1335, Sept. 2005.

[3] V. Sukharev, A. Kteyan, E. Zschech, and W. D. Nix, "Microstructure Effect on EM-Induced Degradations in Dual Inlaid Copper Interconnects," *IEEE Transactions on Device and Materials Reliability*, vol. 9, no. 1, pp. 87–97, 2009.

[4] "International technology roadmap for semiconductors (ITRS), 2012 update," 2012. `http://public.itrs.net`.

[5] S. R. Nassif and J. N. Kozhaya, "Fast power grid simulation," in *Proc. Design Automation Conf. (DAC)*, pp. 156–161, 2000.

[6] J. M. Wang and T. V. Nguyen, "Extended Krylov subspace method for reduced order analysis of linear circuit with multiple sources," in *Proc. Design Automation Conf. (DAC)*, pp. 247–252, 2000.

[7] H. F. Qian, S. R. Nassif, and S. S. Sapatnekar, "Random walks in a supply network," in *Proc. Design Automation Conf. (DAC)*, pp. 93–98, 2003.

[8] T. Chen and C. C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative method," in *Proc. Design Automation Conf. (DAC)*, pp. 559–562, 2001.

[9] Y. Lee, Y. Cao, T. Chen, J. Wang, and C. Chen, "HiPRIME: Hierarchical and passivity preserved interconnect macromodeling engine for RLKC power delivery," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 797–806, 2005.

[10] NVIDIA Corporation, 2011. http://www.nvidia.com.

[11] "NVIDIA Tesla's Servers and Workstations." `http://www.nvidia.com/object/tesla-servers.html`.

[12] NVIDIA Corporation, "CUBLAS library v5.0." `https://developer.nvidia.com/cublas`.

[13] A. M. Sridhar, A. Vincenzi, *et al.*, "3D-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 463–470, IEEE Press, 2010.

[14] L. Ren, X. Chen, Y. Wang, C. Zhang, and H. Yang, "Sparse LU factorization for parallel circuit simulation on GPU," in *Proc. Design Automation Conf. (DAC)*, pp. 1125–1130, 2012.

[15] Z. Feng, Z. Zeng, and P. Li, "Parallel on-chip power distribution network analysis on multi-core-multi-GPU platforms," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 10, pp. 1823–1836, 2011.

[16] K. Daloukas, N. Evmorfopoulos, G. Drasidis, M. Tsiampas, P. Tsompanopoulou, and G. Stamoulis, "Fast transform-based preconditioners for large-scale power grid analysis on massively parallel architectures," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 384–391, November 2012.

[17] J. Wang, "Deterministic random walk preconditioning for power grid analysis," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 392–398, November 2012.

[18] T. Yu, Z. Xiao, and M. D. F. Wong, "Efficient parallel power grid analysis via additive schwarz method," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 399–406, 2012.

[19] S.-H. Weng, Q. Chen, N. Wong, and C.-K. Cheng, "Circuit simulation via matrix exponential method for stiffness handling and parallel processing," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 407–414, November 2012.

[20] "http://crd.lbl.gov/ xiaoye/superlu/."

[21] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse gaussian elimination," *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 4, pp. 915–952, 1999.

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

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

[24] X. Chen, W. Wu, Y. Wang, H. Yu, and H. Yang, "An escheduler-based data dependence analysis and task scheduling for parallel circuit simulation," *IEEE Trans. on Circuits and Systems II: Express Briefs*, pp. 702–706, October 2011.

[25] X. Chen, Y. Wang, and H. Yang, "NICSLU: An adaptive sparse matrix solver for parallel circuit simulation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, pp. 261–274, February 2013.

[26] N. Galoppo, N. 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*, p. 3, 2005.

[27] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Computing*, vol. 36, pp. 232–240, June 2010.

[28] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense Linear Algebra Solvers for Multicore with GPU Accelerators," in *Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–8, IEEE, 2010.

[29] Z. Feng and P. Li, "Multigrid on GPU: tackling power grid analysis on parallel SIMT platforms," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 647–654, 2008.

[30] M. Wang, H. Klie, *et al.*, "Solving sparse linear systems on NVIDIA Tesla GPUs," in *Proc of the 9th Intl. Conf. on Computational Science*, pp. 864–873, 2009.

[31] L. Buatois, G. Caumon, and B. Levy, "Concurrent number cruncher: a GPU implementation of a general sparse linear solver," *Int. J. Parallel Emerg. Distrib. Syst.*, vol. 24, no. 3, pp. 205–223, 2009.

[32] L. Ziane Khodja, R. Couturier, A. Giersch, and J. M. Bahi, "Parallel sparse linear solver with GMRES method using minimization techniques of communications for gpu clusters," *J. Supercomput.*, vol. 69, pp. 200–224, July 2014.

[33] J. M. Bahi, R. Couturier, and L. Z. Khodja, "Parallel GMRES implementation for solving sparse linear systems on GPU clusters," in *Proceedings of the 19th High Performance Computing Symposia*, HPC '11, (San Diego, CA, USA), pp. 12–19, Society for Computer Simulation International, 2011.

[34] J. Bahi, R. Couturier, and L. Khodja, "Parallel sparse linear solver GMRES for GPU clusters with compression of exchanged data," in *Euro-Par 2011: Parallel Processing Workshops* (M. Alexander, P. DAmbra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. Traff, G. Valle, and J. Weidendorfer, eds.), vol. 7155 of *Lecture Notes in Computer Science*, pp. 471–480, Springer Berlin Heidelberg, 2012.

[35] Y. Deng, B. Wang, and S. Mu, "Taming irregular EDA applications on GPUs," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pp. 539–546, 2009.

[36] X. Liu, K. Zhai, Z. Liu, K. He, S. X.-D. Tan, and W. Yu, "Parallel thermal analysis of 3D integrated circuits with liquid cooling on CPU-GPU platforms," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, pp. 575–579, March 2015.

[37] M. Tehranipoor, H. Salmani, and X. Zhang, *Integrated Circuit Authentication*. Springer, 2014.

[38] "U.S. Department of Commerce, Defense Industrial Base Assessment: Counterfeit Electronics,," Jan. 2010.

[39] L. Kessler and T. Sharpe, "Faked Parts Detection," 2010. `http://smtcorp.com/ext/manual/united-el-article-2012-08-22.html`.

[40] M. Times, "Officials: Fake Electronics Ticking Time Bombs." `http://www.militarytimes.com/news/2011/11/ap-fake-electronics-ticking-time-bomb-110811/`.

[41] T. Semiconductor, "3D-ICs and Integrated Circuit Security," 2008. `http://www.tezzaron.com/about/papers/3D-ICs_and_Integrated_Circuit_Security.pdf`.

[42] X. Zhang, N. Tuzzio, and M. Tehranipoor, "Identification of recovered ICs using fingerprints from a light-weight on-chip sensor," in *Proc. Design Automation Conf. (DAC)*, 2012.

[43] X. Zhang and M. Tehranipoor, "Path delay Fingerprinting for Identification of Recovered ICs," in *IEEE Int. Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems(DFT)*, 2012.

[44] J. Villasenor and M. Tehranipoor, "Are you sure its new? the hidden dangers of recycled electronics components," in *IEEE Spectrum*, 2012.

[45] "Failure Mechanisms and Models for Semiconductor Devices." In JEDEC Publication JEP122-A, Jedec Solid State Technolgy Association, 2002.

[46] X. Huang, T. Yu, V. Sukharev, and S. X.-D. Tan, "Physics-based electromigration assessment for power grid networks," in *Proc. Design Automation Conf. (DAC)*, June 2014.

[47] Intel Corporation, "Intel multi-core processors, making the move to quad-core and beyond (White Paper)," 2006. `http://www.intel.com/multi-core`.

[48] AMD Inc., "Multi-core processors—the next evolution in computing (White Paper)," 2006. `http://multicore.amd.com`.

[49] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. Design Automation Conf. (DAC)*, pp. 746–749, 2007.

[50] D. B. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach, 2ed*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2013.

[51] D. Göddeke, "General-purpose computation using graphics harware." http://www.gpgpu.org/, 2011.

[52] X. Liu, S. X.-D. Tan, and H. Yu, "A gpu-accelerated parallel shooting algorithm for analysis of radio frequency and microwave integrated circuits," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 23, March 2015.

[53] 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*, vol. 36, no. 2-3, pp. 321–341, 2007.

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

[55] T. Davis, "The University of Florida sparse matrix collection." http://www.cise.ufl.edu/research/sparse/.

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

[57] I. S. Duff and J. Koster, "The design and use of algorithms for permuting large entries to the diagonal of sparse matrices," *SIAM J. Matrix Anal and Applics*, no. 4, pp. 889–901, 1997.

[58] P. R. Amestoy, Enseeiht-Irit, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minumum degree ordering algorithm," *ACM Trans. Mathematical Software*, pp. 381–388, September 2004.

[59] L. de Soras, "Denormal numbers in floating point signal processing applications," 2002.

[60] X. Chen, L. Ren, Y. Wang, and H. 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.

[61] Z. Liu, S. X.-D. Tan, H. Wang, Y. Hua, and A. Gupta, "Compact thermal modeling for packaged microprocessor design with practical power maps," *Integration, the VLSI Journal*, vol. 47, January 2014. in press, online access: http://www.sciencedirect.com/science/article/pii/S0167926013000412.

[62] Z. Liu, S. Swarup, S. X.-D. Tan, H. Chen, and H. Wang, "Compact lateral thermal resistance model of TSVs for fast finite-difference based thermal analysis of 3D stacked ICs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, Oct 2014. in press.

[63] X. Liu, H. Wang, and S. X.-D. Tan, "Parallel power grid analysis using preconditioned GMRES solvers on CPU-GPU platforms," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 561–568, Nov. 2013.

[64] Y. Saad, *Iterative methods for linear systems*. PWS publishing, 2000.

[65] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. on Sci and Sta. Comp.*, pp. 856–869, 1986.

[66] M. Naumov, "Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU," NVIDIA Technical Report NVR-2011-001, NVIDIA Corp., June 2011.

[67] NVIDIA Corporation, "CUSPARSE library v5.0," October 2012. `http://developer.nvidia.com/cuSPARSE`.

[68] R. Li and Y. Saad, "GPU-accelerated preconditioned iterative linear solvers," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 443–466, 2010.

[69] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec. 2008.

[70] "Cusp-library – generic parallel algoritgms for sparse matrix and graph computations." `http://code.google.com/p/cusp-library`.

[71] "IBM power grid benchmarks." `http://dropzone.tamu.edu/~pli/PGBench/`.

[72] J. Mayer, "ILU++ package." `www.iluplusplus.de/`.

[73] J. Mayer, "A multilevel Crout ILU preconditioner with pivoting and row permutation," *Numer. Linear Algebra Appl.*, vol. 14, pp. 771–789, December 2007.

[74] M. Hauschildt, C. Hennesthal, G. Talut, O. Aubel, M. Gall, K. B. Yeap, and E. Zschech, "Electromigration Early Failure Void Nucleation and Growth Phenomena in Cu And Cu(Mn) Interconnects," in *IEEE International Reliability Physics Symposium (IRPS)*, pp. 2C.1.1–2C.1.6, 2013.

[75] M. A. Korhonen, P. Borgesen, K. N. Tu, and C. Y. Li, "Stress Evolution Due to Electromigration in Confined Metal Lines," *Journal of Applied Physics*, vol. 73, no. 8, pp. 3790–3799, 1993.

[76] J. J. Clement, "Reliability Analysis for Encapsulated Interconnect Lines under DC and Pulsed DC Current Using A Continuum Electromigration Transport Model," *Journal of Applied Physics*, vol. 82, no. 12, pp. 5991–6000, 1997.

[77] M. E. Sarychev, Y. V. Zhitnikov, L. Borucki, C.-L. Liu, and T. M. Makhviladze, "General Model for Mechanical Stress Evolution during Electromigration," *Journal of Applied Physics*, vol. 86, no. 6, pp. 3068–3075, 1999.

[78] V. Sukharev, "Beyond Black's Equation: Full-Chip EM/SM Assessment in 3D IC Stack," *Microelectronic Engineering*, vol. 120, pp. 99–105, 2014.

[79] V. Sukharev, A. Kteyan, J.-H. Choy, H. Hovsepyan, A. Markosian, E. Zschech, and R. Huebner, "Multi-scale Simulation Methodology for Stress Assessment in 3D IC: Effect of Die Stacking on Device Performance," *Journal of Electronic Testing*, vol. 28, pp. 63–72, Nov. 2011.

[80] Z. Suo, *Reliability of Interconnect Structures*, vol. 8 of *Comprehensive Structural Integrity*. Amsterdam: Elsevier, 2003.

[81] J. R. Black, "Electromigration-A Brief Survey and Some Recent Results," *IEEE Trans. on Electron Devices*, vol. 16, no. 4, pp. 338–347, 1969.

[82] D. Chardonnal, "Impacts of counterfeiting and piracy to reach US$1.7 trillion by 2015," 2011.

[83] Trust-HUB. `http://trust-hub.org/home`.

[84] U. Guin, D. DiMase, and M. Tehranipoor, "Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead," *Journal of Electronic Testing*, vol. 30, pp. 9–23, Feb. 2014.

[85] K. He, X. Huang, and S. X.-D. Tan, "EM-Based on-chip aging sensor for detection and prevention of counterfeit and recycled ICs," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, Nov. 2015.

[86] F. Koushanfar, G. Qu, and M. Potkonjak, *Intellectual Property Metering*. Springer, 2001.

[87] F. Koushanfar and G. Qu, "Hardware Metering," in *Proc. Design Automation Conf. (DAC)*, pp. 490–493, 2001.

[88] kilopass White Paper, "Three SoC Application Segments Require Embedded OTP Memory."

[89] S. Rezgui, J. Wang, Y. Sun, D. D'Silva, B. Cronquist, and J. McCollum, "SET characterization and mitigation in RTAX-S antifuse FPGAs," in *IEEE Aerospace Conference*, March 2009.

[90] J. McCollumn, "ASIC versus antifuse FPGA reliability," in *IEEE Aerospace Conference*, March 2009.

[91] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Inproceedings of 16th USENIX security symposium on USENIX security symposium*, pp. 1–16, 2007.

[92] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," vol. 27, pp. 66–75, 2010.