PARALLEL ALGORITHMS FOR TIME AND FREQUENCY DOMAIN CIRCUIT SIMULATION

A Dissertation

by

WEI DONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2009

Major Subject: Computer Engineering

PARALLEL ALGORITHMS FOR TIME AND FREQUENCY DOMAIN CIRCUIT SIMULATION

A Dissertation

by

WEI DONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2009

Major Subject: Computer Engineering

ABSTRACT

Parallel Algorithms for Time and Frequency Domain Circuit Simulation.

(August 2009)

Wei Dong, B.E., Xi'an JiaoTong University;

M.E., Shanghai JiaoTong University

Chair of Advisory Committee: Dr. Peng Li

As a most critical form of pre-silicon verification, transistor-level circuit simulation is an indispensable step before committing to an expensive manufacturing process. However, considering the nature of circuit simulation, it can be computationally expensive, especially for ever-larger transistor circuits with more complex device models. Therefore, it is becoming increasingly desirable to accelerate circuit simulation. On the other hand, the emergence of multi-core machines offers a promising solution to circuit simulation besides the known application of distributed-memory clustered computing platforms, which provides abundant hardware computing resources. This research addresses the limitations of traditional serial circuit simulations and proposes new techniques for both time-domain and frequency-domain parallel circuit simulations.

For time-domain simulation, this dissertation presents a parallel transient simulation methodology. This new approach, called WavePipe, exploits coarse-grained application-level parallelism by simultaneously computing circuit solutions at multiple adjacent time points in a way resembling hardware pipelining. There are two embodiments in WavePipe: backward and forward pipelining schemes. While the former creates independent computing tasks that contribute to a larger future time step, the latter performs predictive computing along the forward direction. Unlike existing relaxation methods, WavePipe facilitates parallel circuit simulation without jeopardizing convergence and accuracy. As a coarse-grained parallel approach, it requires low parallel programming effort, furthermore it creates new avenues to have a full utilization of increasingly parallel hardware by going beyond conventional finer grained parallel device model evaluation and matrix solutions.

This dissertation also exploits the recently developed explicit telescopic projective integration method for efficient parallel transient circuit simulation by addressing the stability limitation of explicit numerical integration. The new method allows the effective time step controlled by accuracy requirement instead of stability limitation. Therefore, it not only leads to noticeable efficiency improvement, but also lends itself to straightforward parallelization due to its explicit nature.

For frequency-domain simulation, this dissertation presents a parallel harmonic balance approach, applicable to the steady-state and envelope-following analyses of both driven and autonomous circuits. The new approach is centered on a naturally-parallelizable preconditioning technique that speeds up the core computation in harmonic balance based analysis. The proposed method facilitates parallel computing via the use of domain knowledge and simplifies parallel programming compared with fine-grained strategies. As a result, favorable runtime speedups are achieved.

To my parents and my wife

ACKNOWLEDGMENTS

I wish to express my great thanks to my advisor Dr. Peng Li. He introduced the world of EDA to me. I truly appreciate his continuous financial support and research guidance throughout my Ph.D. Studies. Dr. Li has shared his profound knowledge and professional manner of conducting research. He was so patient in reviewing my drafts of papers and dissertation. He is always there to listen to and give valuable advice for any problems I have. I feel so lucky to be able to study under his guidance and what I learned from him will definitely benefit my whole career.

I would like to thank Dr. Sunil P. Khatri, Dr. Jose Silva-Martinez and Dr. Duncan M. (Hank) Walker for being my committee members, and for their patience, time, and valuable suggestions for my Ph.D. program.

My sincere thanks to all the members in our research group for their help and friendship. Special thanks to Zhuo Feng, Guo Yu and Xiaoji Ye for their knowledge, help, and valuable discussion.

I would like to acknowledge the financial support from NSF, SRC, C2S2 and Texas Analog Center for Excellence for my research.

Finally, I would like to thank my parents and my wife, for their unconditional love, support and trust throughout these years.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1
A. Backgrounds and motivations 1
1. New trend on parallel computing platform for CAD . 1
2. Performance concerns on circuit simulation 3
3. Overview on existing parallel simulation methods . . . 4
B. New contributions 5
C. Outline 7

II FUNDAMENTALS OF PARALLEL COMPUTING 9
A. Introduction 9
B. Organization of parallel computing platforms 11
1. Logical organization of parallel platforms 11
2. Physical organization of parallel platforms 12
C. Parallel programming models 13
1. Message passing interface model 14
2. Threaded shared memory programming model 15
3. Comparison of parallel programming models 17
D. Performance metrics 17

III CIRCUIT SIMULATION IN TIME AND FREQUENCY DOMAIN 22
A. Introduction 22
B. Time domain analysis: transient simulation 23
1. Introduction 23
2. Numerical integration methods 25
C. Frequency domain analysis: harmonic balance simulation . 27

IV COARSE-GRAINED WAVEPIPE PARALLEL TRANSIENT SIMULATION 33
A. Introduction 33
B. Backward pipelining 35
1. Variable-step size multi-step methods 36
2. Backward pipelining 38

CHAPTER Page

C. Forward pipelining 42
1. Prediction of time step size 43
2. Accuracy and stability 43
D. Multi-threaded WavePipe and thread scheduling 44
1. Thread scheduling 45
2. Scheduling policies 47
E. Experimental results 48
F. Summary 51

V PARALLEL TRANSIENT SIMULATION BASED ON EXPLICIT INTEGRATION METHOD 53

A. Introduction 53
B. Principle of telescopic projective integration 54
C. Stable explicit numerical integration for circuit simulation 58
1. Stability of the standard Forward Euler 60
2. Stability of projective integration 60
3. Stability of telescopic projective integration 63
4. Parallel implementation 64
D. Experimental results 67
1. Accuracy and efficiency 67
2. Serial and parallel simulation 70
E. Summary 71

VI PARALLEL HARMONIC BALANCE SIMULATION 73

A. Introduction 73
B. Proposed parallel HB analysis 75
1. Basic ideas of parallel hierarchical preconditioning . . 78
2. Analysis of runtime complexity and parallel efficiency 82
3. Processing element allocation 83
C. Extensions to parallel autonomous circuit and envelope-following analyses 87
1. Parallel steady-state analysis of autonomous circuits . 88
2. Parallel envelope-following analysis 91
D. Implementation issues 95
E. Experimental results 97
1. Performance of driven circuit simulation 99
2. Performance of autonomous circuit simulation 104
3. Performance of envelope-following simulation 105

CHAPTER Page

F. Summary 110

VII APPLICATION OF PARALLEL HARMONIC BALANCE SIMULATION TO MASSIVE CLOCK MESHES 111

A. Introduction 111
B. Computation of mesh transfer functions 113
C. Clock mesh analysis via harmonic balance 115
D. Experimental results 116
E. Summary 118

VIII CONCLUSIONS AND FUTURE WORK 121

A. Conclusions 121
B. Future work 122

REFERENCES 125

VITA 138

LIST OF TABLES

TABLE Page

I Comparisons of parallel programming techniques. 17

II Performance metrics for parallel systems. 18

III Characteristics of the numerical integration methods. 26

IV Statistics of test circuits and serial BE. 48

V Runtime speedups of 2-threaded coarse-grained WavePipe schemes. . 49

VI Runtime speedups of 3-threaded coarse-grained WavePipe schemes. . 49

VII Runtime speedups of 4-threaded coarse-grained WavePipe schemes. . 50

VIII Statistics of the transient simulations on serial and parallel platforms. 71

IX Descriptions of the driven circuits. 99

X Statistics of the serial HB simulations for the driven circuits. 100

XI Runtime statistics of three key steps and their parallelization on the 3-CPU platform. 100

XII Statistics of the parallel HB simulations on the 3-CPU / 9-CPU platforms for the driven circuits. 101

XIII Descriptions of the autonomous circuits. 104

XIV Statistics of the HB simulations on serial platform for the oscillators. 104

XV Statistics of the HB simulations on parallel platforms for the oscillators. 105

XVI Statistics of the envelope-following simulations on serial platform. . . 109

TABLE Page

XVII Statistics of the envelope-following simulations on parallel platforms. 110

XVIII Comparison for full transient simulation and proposed parallel HB simulation. 119

LIST OF FIGURES

FIGURE Page

1 Different multicore architectures. 9

2 Serial computing and parallel computing. 10

3 Comparison of SIMD and MIMD architectures. 12

4 Two types of logical organization of parallel platforms. 13

5 Two types of physical organization of parallel platforms. 14

6 Mechanism of MPI and thread based parallelization. 15

7 A basic flow of transient simulation. 24

8 BD preconditioner and hierarchical preconiditioner. 31

9 Generation of hierarchial preconditioner. 31

10 Data dependency in a) one-step, and b) multi-step (2-step) numerical integration. 35

11 Parallel double-threaded backward pipelining: a) an naïve approach, and b) the proposed backward pipelining. 39

12 Double-threaded forward pipelining. 42

13 Fine and coarse grained inter-thread communications to guarantee the accuracy of the parallel forward scheme. 45

14 Four-thread waveform pipelining. 46

15 4T waveform pipelining: a) without revoking of forward pipelining and b) with revoking of forward pipelining. 47

16 Speedups of various WavePipe schemes. 50

FIGURE Page

17 Realtime thread profiling of the 3T one-forward-one-backward waveform pipelining. 51

18 Comparison between WavePipe and low-level parallel model evaluation/matrix solving: double-balanced mixer. 52

19 Comparison between WavePipe and low-level parallel model evaluation/matrix solving: RLC mesh. 52

20 Distribution of the eigenvalues with a single gap. 55

21 Distribution of the eigenvalues with multiple gaps. 57

22 Telescopic projective framework. 57

23 Proposed stable explicit numerical integration for circuit simulation. 59

24 Two circuit nodes: (a) without coupling, and (b) with coupling. . . 65

25 Parallel simulation framework. 67

26 Stiff RC circuit 1. 68

27 Input waveform. 68

28 Transient simulation for circuit 1. 69

29 Stiff RC circuit 2. 70

30 Transient simulation for circuit 2. 71

31 A basic flow for HB analysis. 75

32 Task dependency of the operations in each Newton iteration for HB analysis. 76

33 Parallelization of FFT/IFFT operations. 77

34 Tree-like problem decomposition for the hierarchical preconditioner. 81

FIGURE Page

35 The task-dependency graph of the hierarchical preconditioner. . . . 84

36 Allocation of processing elements for hierarchical preconditioning. . . 85

37 Size-dependent PE allocation for a three-level preconditioner. 87

38 Voltage probe. 88

39 Parallelizable autonomous circuit HB analysis. 89

40 Partitioning of the Jacobian of autonomous circuits. 92

41 Envelope-following analysis. 93

42 The algorithm flow of parallel envelope-following analysis. 96

43 Non-blocking data transfers. 98

44 The runtime speedups of the parallel HB with hierarchical preconditioning vs. the number of the processors. 102

45 Comparison of shared-memory and distributed-memory implementations. 103

46 A schematic of power amplifier. 105

47 A schematic of double-balanced mixer. 106

48 Transient simulation of the power amplifier. 107

49 Envelope-following simulation of the power amplifier. 108

50 Transient simulation of the double-balanced mixer. 108

51 Envelope-following simulation of the double-balanced mixer. 109

52 Non-tree clock distributions. 112

53 Non-zero patterns in the Jacobian matrix. 115

54 Comparison between transient simulation & hierarchical HB simulation for mesh1 [full waveform view]. 117

FIGURE Page

55 Comparison between transient simulation & hierarchical HB simulation for mesh1 [zoomed-in view]. 118

56 Comparison between transient simulation & hierarchical HB simulation for mesh2 [full waveform view]. 119

57 Comparison between transient simulation & hierarchical HB simulation for mesh2 [zoomed-in view]. 120

58 Combinations of multiple parallelization techniques. 123

CHAPTER I

INTRODUCTION

A. Backgrounds and motivations

1. New trend on parallel computing platform for CAD

The most recent advances in microprocessor design involve putting multiple processors on a single computer chip. In this new architecture, all processors can execute instructions independently and simultaneously with full function units like ALU, FPU, Caches, etc in each core. Therefore, multicore is possible to extract more performance from the unit chip area.

These multicore designs are completely replacing the traditional single core designs that have been the foundation of computers. All computer chip manufacturers, such as IBM [1], Sun [2], Intel [3] and AMD [4], have changed their chip product line from single core processor production to multicore processor production. Facing the physical limits of semiconductor-based micro-electronics, it seems that the migration to multicore is the only way to maintain Moore's Law due to the following considerations [5]:

(1) The limitation on transistors size: Due to being close to the physical limits of semiconductor-based micro electronics, IC manufacturing procedure suffers from difficulties such as variability and lithography issues, which are greatly limited to further scaling. It is also expected that the quantum behavior of electrons will gradually show up to affect the operations with the shrinking transistor size. Therefore, the traditional way to use higher CPU clock frequency on new products by reducing

The journal model is *IEEE Transactions on Automatic Control.*

the size of transistors, which can reduce the distances between the transistors and decrease transistor switching times, doesn't work well any more.

(2) The limitation on power consumption: Even without considering the size limitation on transistors, the heat dissipation and power consumption are also big barriers for high frequency operations. Because the frequency is related to the supply voltage of chips and the power is related to the square of the supply voltage, the heat dissipation would increase much higher with the increase of clock frequency. In another word, the traditional way to improve the processor performance by increasing the clock frequency cannot continuously work well due to power consumption. To address this problem, instead of using one high-frequency processor with frequency magnitude of NfHz, in a multicore platform, N low-frequency processors with the frequency magnitudes of fHz are simultaneously employed to have the similar performance, while the power consumption of the latter is only $1/N$ of that of the former.

With chip vendors pushing the envelope on the number of cores on a single chip, more computing power is in the hands of consumers than ever before. This trend not only significantly affects the personal computer platform but also the distributed cluster computer networks equipped with multicore chips. However, the primary problem is that most existing software has not been designed to take advantage of parallel hardware [5], especially in computationally-intensive application areas such as computer-aided design (CAD). The emergence of multiprocessors on a single chip brings both challenges and opportunities to CAD [6]. On one hand, traditional serial CAD software can not make full use of highly parallel machines and has to be re-architected to discover and express high degrees of parallelism; on the other hand, it is expected that future performance increases will be provided greatly through increased on-chip parallelism.

It is also interesting to exploit the parallelism on distributed-memory computing platforms. This is due to the following considerations. Firstly, the distributed-memory computing platforms are still one of main existing platforms for high-performance computing, especially when a large number of processing elements are required for massively parallelization. Secondly, the emergence of multicore (or even manycore in future) may not substitute the distributed-memory computing platform. On the contrary, the distributed-memory computing platform will adopt the multicore machines to construct a clustered of multicore network. And the hybrid computing platform will be widely used for parallel computing.

To sum up, it is one of promising way for CAD software to make computational speedups by targeting to parallelism instead of continual improvements in single thread performance.

2. Performance concerns on circuit simulation

Considering a basic CAD software system for integrated circuit (IC) design, circuit simulation, as the most critical forms of pre-silicon verification, is an indispensable step before committing to an expensive manufacturing process. Consequently, the performance of circuit simulation is critical to the success of IC design, especially when viewed from an economic return perspective.

Different methods have been employed to circuit simulation [7–9], like time-domain integration [10–19], harmonic balance (HB) in frequency domain [20–31], or the more recent mixed time-frequency approaches [32–37]. The particular choice of simulation method depends on the type of circuit and application. However, no matter which kind of simulation method is used, accelerating circuit simulation is always desirable.

Devoting to the speedup of circuit simulation, much research has been geared toward the development of more efficient algorithms and more effective implementations. Most of such efforts to accelerate simulations are achieved by elimination of redundant calculations, simplification of models, and replacement of time-consuming accurate algorithms by less accurate ones [38–40]. To further improve the performance of circuit simulation, especially with the trend of hardware migration from serial processing to parallel processing, it is promising to develop more efficient parallel data-structures, and advanced programming techniques.

3. Overview on existing parallel simulation methods

Parallel computing techniques have already been in the scope of research interests for circuit simulations for decades. With the recent trend of the migration to multicore, the requirement of parallel computing research is becoming more demanding. This is partly because the traditional parallel machines were extraordinarily expensive and the only a few circuit designers had the privilege to use them. And it is also because many circuit simulation problems nowadays are becoming so large and complex that they cannot be solved in a sequential way within a reasonable time limit.

As one of the most representative simulation methods, time-domain transient simulation is indispensable for a broad range of designs, especially for analog and radio-frequency (RF) circuits [19]. The time-consuming nature of transient analysis often makes it a significant bottleneck, necessitating its parallelization. There exist a number of parallel simulation approaches, majority of which are fine grained in nature. It is known that the most commonly-used technique is to parallelize the key steps in an existing simulation algorithm, e.g. device model evaluation and matrix solve. However, the efficiency of parallel matrix solvers can deteriorate fairly quickly as the number of processor cores increases. On supercomputers and computer clusters, waveform relaxation and other nonlinear relaxation methods have been proposed for parallel circuit simulation [38,39,41]. However, these methods are not widely used for robust general circuit simulation due to limited convergence properties. The efficiency of the domain decomposition approach in [42, 43] is strongly application dependent, leading to limited applicability. Furthermore, the two approaches above require fine-grained parallel programming, hence high implementation and debugging effort.

Being a counterpart of time-domain transient simulation method, HB simulation is a frequency-domain steady-state simulation method, which can directly obtain the periodic or quasi-periodic steady-state solution waveforms. Since the problem is formulated based on a set of harmonics, the problem size is much larger than that of time-domain transient analysis. To facilitate parallel HB analysis, various techniques have been proposed (e.g. [44–47]). In [44], Rhodes and Perlman propose a method to partition a circuit into linear and nonlinear portions so that the solution of the linear portion is parallelized based on the assumption that it dominates the overall runtime. However, this assumption is not always the case. The authors of [45, 46] extend the work in [44] by exposing the potential parallelism in the form of a directed acyclic graph (DAG). Several application-domain specific methods for allocation and scheduling are discussed. In [47], an implementation of HB analysis on shared memory multicomputers has been reported, where the parallel task allocation and scheduling are applied to device model evaluation, the matrix-vector product and the standard block-diagonal (BD) preconditioning. However, it is a fine-grained parallelization and the block-diagonal preconditioning is only parallelizable in a per-frequency basis.

B. New contributions

By addressing the limitations of existing simulation techniques above, this research work is focused on new parallel simulation algorithms and their efficient implementations on shared-memory and distributed-memory platforms.

(1) A parallel transient simulation methodology is proposed for general analog and digital ICs. This new approach, *Waveform Pipelining* (abbreviated as *WavePipe*), exploits coarse-grained application-level parallelism by simultaneously computing circuit solutions at multiple adjacent time points in a way resembling hardware pipelining. There are two embodiments in *WavePipe*: backward and forward pipelining schemes. While the former creates independent computing tasks that contribute to a larger future time step by moving backwards in time, the latter performs predictive computing along the forward direction of the time axis. Unlike existing relaxation methods, *WavePipe* facilitates parallel circuit simulation without jeopardying convergence and accuracy. As a coarse-grained parallel approach, *WavePipe* not only requires low parallel programming effort, more importantly, it creates new avenues to fully utilize increasingly parallel hardware by going beyond the conventional finer grained parallel device model evaluation and matrix solve.

(2) The recently developed explicit telescopic projective numerical integration method is exploited for efficient parallel transient simulation. By addressing the well-known stability limitation of explicit numerical integration with a rigorous theoretical basis, the use of telescopic projective integration makes the *effective* time step no longer be limited by the smallest time constant in the circuit while ensuring stability. This new stable explicit numerical integration approach not only leads to noticeable efficiency improvement in circuit simulation, but also lends itself to straightforward parallelization due to its explicit nature.

(3) A parallel HB simulation approach is proposed, which is applicable to the steady-state and envelope-following analyses of both driven and autonomous circuits. This approach is centered on a naturally-parallelizable preconditioning technique that speeds up the core computation in HB based analysis. As a coarse-grained parallel approach by algorithm construction, the proposed method facilitates parallel comput-

ing via the use of domain knowledge and simplifies parallel programming compared with fine-grained strategies. The proposed parallel preconditioning technique can be combined with more conventional parallel approaches such as parallel device model evaluation, parallel fast fourier transform (FFT) and parallel matrix-vector product to further improve runtime efficiency.

C. Outline

The following chapters of the dissertation are organized as follows. Firstly, the relevant backgrounds and the preliminary knowledge about the organizations of parallel computing platforms, parallel programming models and performance metrics are demonstrated in Chapter II. Then, in Chapter III, the basic ideas, concepts and principles about circuit simulations in time domain and frequency domain are introduced. After these two preliminary chapters, the parallel time-domain transient simulation techniques and the parallel frequency-domain HB-based simulation techniques are discussed in detail. In Chapter IV, a coarse-grained WavePipe parallel transient simulation technique is proposed [48]. To explain this technique, in this chapter, the principle of two basic pipelining mechanisms (backward pipelining and forward pipelining) are first demonstrated. And then the multithreaded parallel idea and the relevant scheduling technique are discussed and validated by the experimental results. Following this chapter, a stable explicit telescopic projective integration method is exploited so that a parallelizable transient simulation technique based on telescopic projective integration method is proposed and validated by the theoretical stability analysis and by the experimental results. In the next two chapters, the research topic is moved to parallel HB simulation and its application. In Chapter VI, a parallel framework for a HB simulation using a parallelizable hierarchical preconditioning technique is proposed [49, 50]. Firstly, the illustration is focused on the standard driven-circuit HB simulation problem. The principle of parallelizable hierarchical preconditioning technique is explained in detail within this scope. Then, the similar principle is generalized to address the autonomous circuit HB simulation problem and the HB-based envelope-following analysis. In this chapter, not only the multithread-based implementation but also the MPI-based implementation are involved in the experiments to validate the proposed parallel HB simulation framework. In Chapter VII, an application of parallel HB framework to massive clock meshes is discussed [51]. Finally, the conclusions are given and the future works are suggested in Chapter VIII.

CHAPTER II

FUNDAMENTALS OF PARALLEL COMPUTING

A. Introduction

Although the emerging trend of migration to multicore is fairly new in the computer architectures field, the concept of 'multicore', as a popular name for chip multiprocessors (CMPs) or single-chip multiprocessors, has been explored by chip manufacturers since the early 1990s [5]. Nowadays, all processor manufacturers have proposed new multicore products based on their understanding of the concepts of CMPs. As shown in Fig. 1, the two different multicore architectures are shown, where the left one is an Intel Core-2-Due multiprocessor and the right is an AMD Opteron multiprocessor.

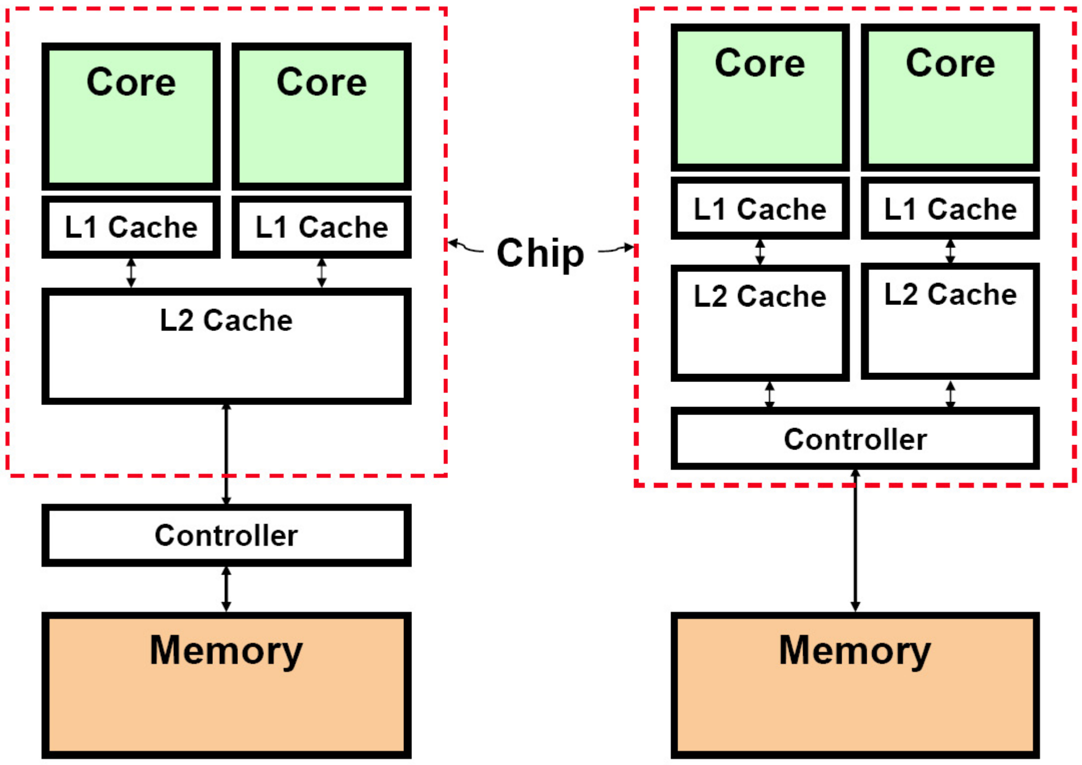

Fig. 1. Different multicore architectures.

An obvious difference in Fig. 1 is using L2 cache in different ways. The benefits of sharing the L2 cache or not are greatly dependent on the overall architecture design and application emphasis. Although, new multicore architectures vary from vendor to vendor and have different features in design, which may lead to significant performance differences, the theoretical model and the analysis method for parallel computing on them are in common and similar to those applied to the distributed clusters of parallel computing platforms.

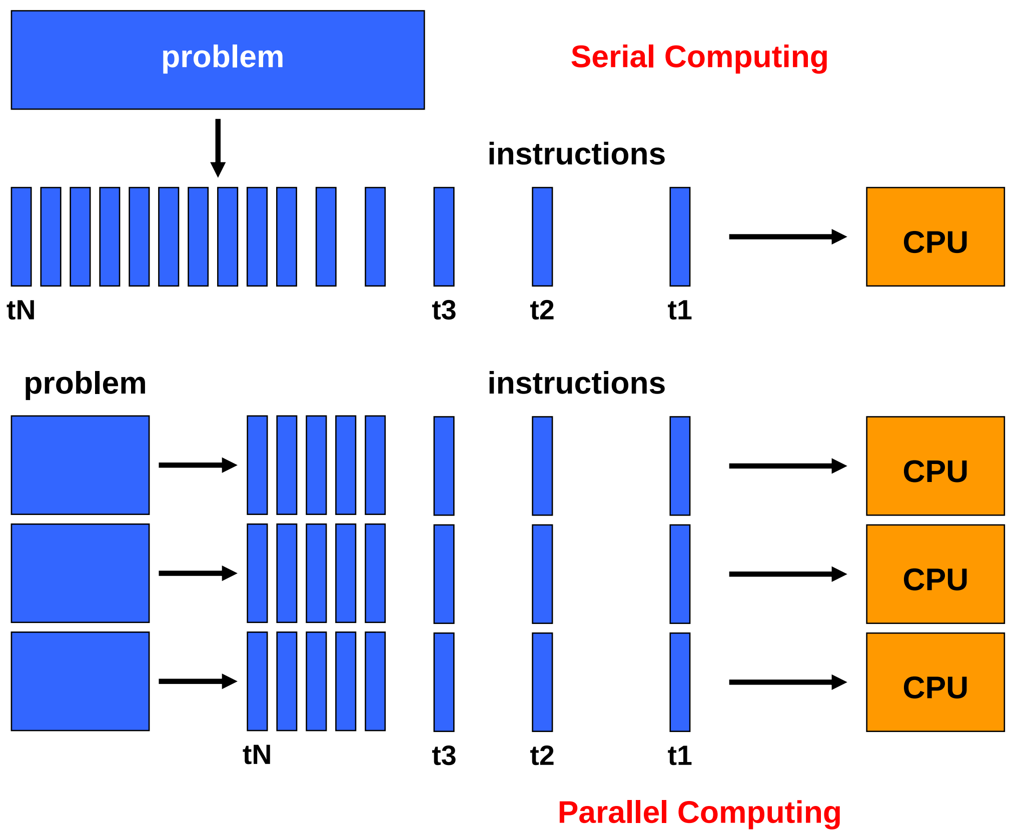

Fig. 2. Serial computing and parallel computing.

B. Organization of parallel computing platforms

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem as shown in Fig.2. Correspondingly, parallel programming requires suitable computing platforms, which are critical for performance oriented and portable parallel programming. A dichotomy is employed based on the logical and physical organization of parallel platforms [52]. The logical organization refers to a programmer's view of the platform while the physical organization refers to the actual hardware organization of the platform.

1. Logical organization of parallel platforms

From a programmer's perspective, there are two critical components of parallel computing, which are the control structure and communication model.

About the control structure, it is known that processing units in parallel computers either operate under the centralized control of a single control unit or work independently. In the architecture referred to as single instruction stream and multiple data stream (SIMD), a single control unit dispatches instructions to each processing unit to process multiple data concurrently. In contrast to SIMD, computers in which each processing element is capable of executing a different program independent of the other processing elements are called multiple instruction stream and multiple data stream (MIMD). In Fig. 3, a comparison between SIMD and MIMD is shown.

There are two primary forms of data exchange between parallel tasks based on different communication models as shown in 4. One form is accessing a shared data space and another form is exchanging messages. In the former form, processors interact by modifying data objects stored in the shared-address-space. In the latter form, interactions between processes running on different nodes must be accomplished

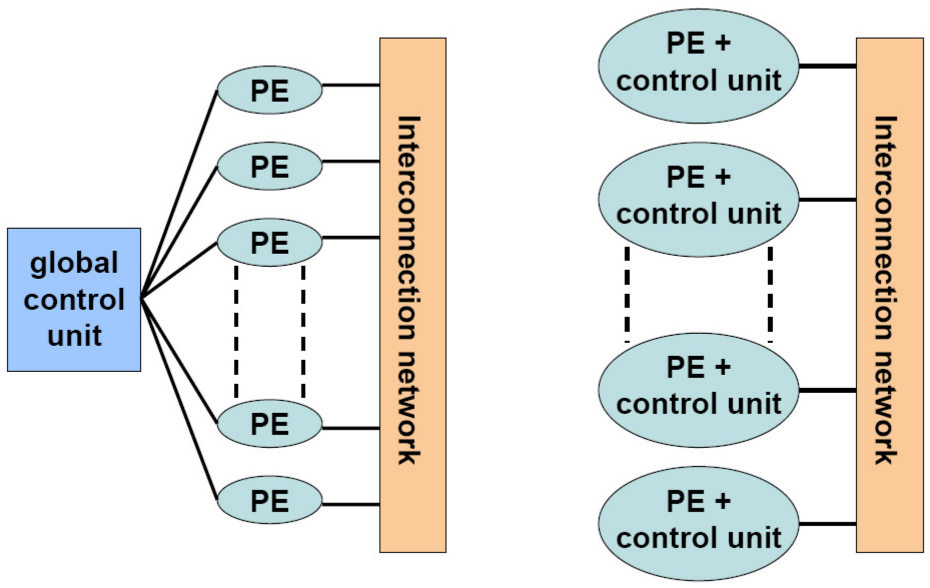

Fig. 3. Comparison of SIMD and MIMD architectures.

using message exchange.

2. Physical organization of parallel platforms

Considering the physical organization, a parallel computer can be categorized into one of two different types of computing platforms based on the memory hierarchy in the computing platform, which are shared-memory and distributed-memory computing platforms as shown in Fig. 5.

In shared-memory computers, memory is physically shared among various processors, allowing processors communicate through variables stored in a shared address space. While in distributed-memory computers, different segments of the memory are physically associated with different processing elements. Processors communicate with each other over the network. It deserves mentioning that there are differences between the concept of shared-address-space communication model and that of shared-memory computers, though the shared-address-space communication mecha-

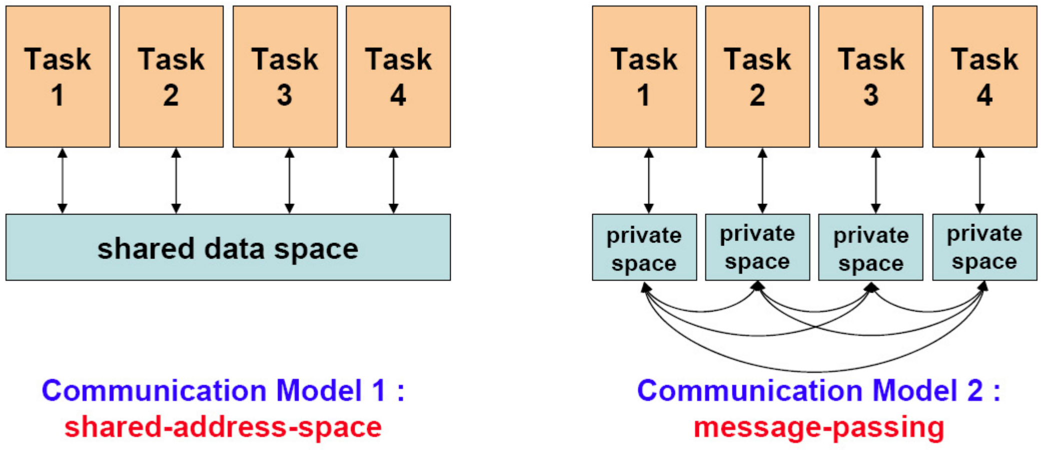

Fig. 4. Two types of logical organization of parallel platforms.

nism is widely used in the shared-memory machines and the message-passing communication mechanism is usually used in the distributed-memory machines. The term shared-memory computer is used for architectures in which the memory is physically shared among various processors as shown in the plot on the left of Fig. 5, which is in contrast to a distributed-memory computer. The dichotomy of shared- versus distributed-memory computers pertains to the physical organization of the machine. Either of these physical models, shared or distributed memory, can present the logical view of a disjoint or shared-address-space platform.

C. Parallel programming models

The architectural differences in parallel computing platforms have implications on how each is programmed. With a shared-memory platform, different processors can access the same variables. This makes referencing data stored in memory similar to traditional single-processor programs, but adds the complexity of shared data integrity. A distributed-memory system introduces a different problem: how to distribute a com-

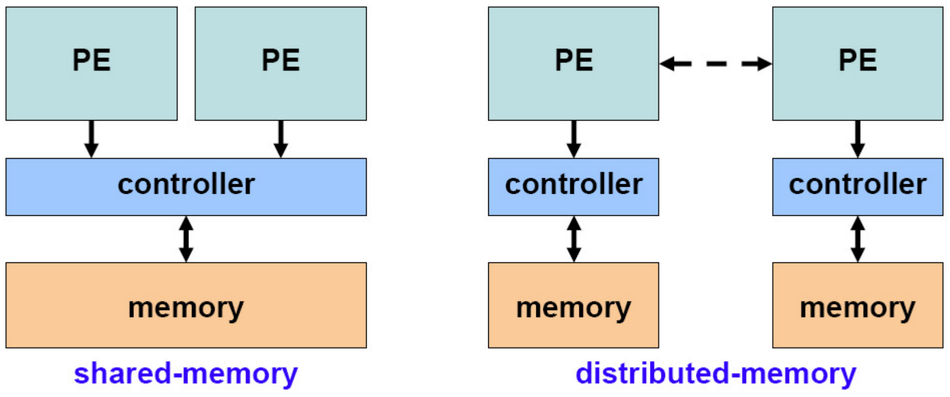

Fig. 5. Two types of physical organization of parallel platforms.

putational task to multiple processors with distinct memory spaces and reassemble the results from each processor into one solution.

Correspondingly, different parallel programming models and techniques are employed for different parallel computing platforms. For example, message passing interface (MPI), is an interface for a set of library functions that processors in a distributed-memory multiprocessor can use to communicate with each other. Pthreads and OpenMP are threaded-based library functions and compiler directives for developing parallel programs on a shared-memory multiprocessor platform. In Fig. 6, the comparison between MPI and threaded-based parallel models is shown.

1. Message passing interface model

Message passing interface (MPI) is a specification for an application programming interface (API) that allows many processors to communicate with one another. It has become a de facto standard for communication among processes that model a parallel program running on a distributed memory system. It deserves mentioning that MPI programs are also able to run on shared memory computers. Designing programs around the MPI model (as opposed to explicit shared memory models)

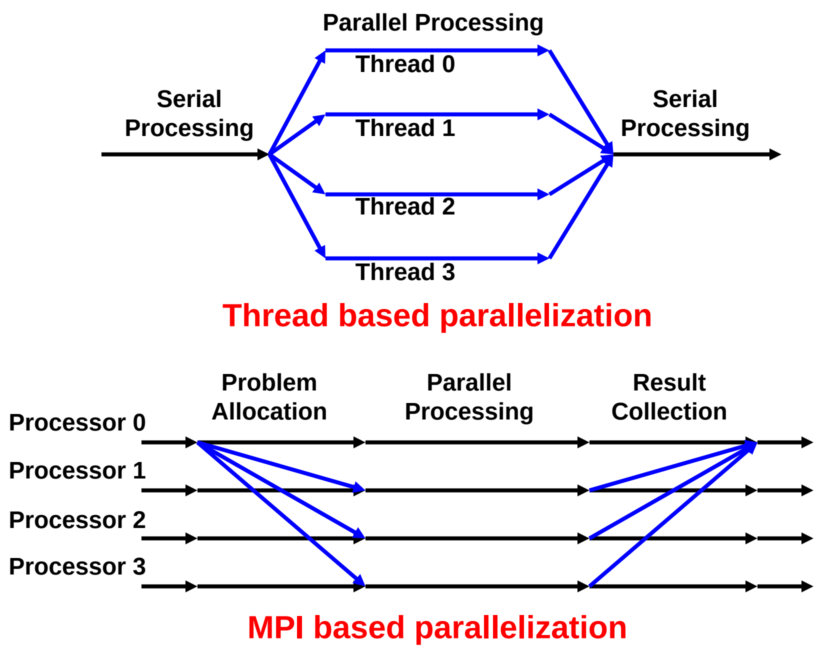

Fig. 6. Mechanism of MPI and thread based parallelization.

has advantages on non-uniform memory access (NUMA) architectures since MPI encourages memory locality.

MPI is a language-independent communications protocol used to program parallel computers. Both point-to-point and collective communication are supported in MPI. Its goals are high performance, scalability, and portability. In all kinds of MPI implementations, MPICH is one of most successful implementations in high-performance computing today [53].

2. Threaded shared memory programming model

Different from MPI programming model, threaded shared-memory programming model is another widely-adopted parallel computing model. In this model, parallel tasks are allocated to each thread and executed simultaneously and independently. Therefore, the parallel task carrier is thread instead of process. Threads are just like processes, only smaller. The idea of threads is that multiple threads of execution can share a lot of resources; for instance, they generally operate in the same address space. Switching from one thread to another is generally cheaper than switching from one process to another. Furthermore, as processes may use a lot of memory, threads may allow substantially more efficient use of memory.

Pthreads is the POSIX standard to provide an application programming interface (API) that supports thread-level parallelization for shared memory platforms. Pthreads specifies the API to handle most actions required by threads. These actions include creating and terminating threads, waiting for threads to complete, and managing the interaction between threads. In the latter category, there exist various locking mechanisms that prevent two threads from trying to modify the same data values simultaneously: mutexes, condition variables, and semaphores. Considering that Pthreads offers a great range of primitive functions that provide fine-grained control over threading operations, in applications in which threads have to be individually managed, Pthreads would be a natural choice.

Since Pthreads provides most extensive controls over thread operations, it is an inherently low-level API that mostly requires multiple steps to perform threading tasks and therefore requires considerable threading-specific code. Moreover, certain decisions, such as the number of threads to use can become hard-coded into the program. Because of the amount of threading code needed to perform straightforward operations, an alternative to Pthreads shall be considered. Compared with Pthreads, the alternative should be a higher-level API for threaded-based parallel programming.

The OpenMP (Open Multi-Processing) is such a good candidate, which consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. By judicious use of these pragmas, a single-threaded program can be made multithreaded without recourse to APIs or environment variables. And it is also convenient to run the same copy of code on different platforms, which possibly have a different number of threads. However, OpenMP lacks of finer-grained control over thread operations, therefore it is difficult to handle some complex parallel operations.

3. Comparison of parallel programming models

In order to show the advantages and disadvantages of different parallel programming models, the comparisons among Pthreads, OpenMP and MPI are listed in Table I.

Table I. Comparisons of parallel programming techniques.

Technique	Pthreads	OpenMP	MPI
Platform	Shared-memory	Shared-memory	Distributed/shared memory
Mechanism	Thread-based memory sharing	Thread-based memory sharing	Message passing
Usage	Library functions	Compiler directives	Library functions
Content	Comprehensive	Lack of finer-grained control	Comprehensive
Programming	Difficult	Easy	Difficult

From Table I, it can be observed that different programming methods have their benefits. Actually, threaded shared memory programming models (such as Pthreads and OpenMP) and message passing programming (MPI) can be considered as complementary programming approaches, and can be used together in applications to take advantage of their benefits together.

D. Performance metrics

When designing and implementing a parallel program, it is important to study the performance of the parallel program to determine the best algorithm, evaluate hardware platforms, and examine the benefits from parallelism. A number of metrics have been used based on the outcome of performance analysis. In Table II, some basic performance metrics are listed [54].

Table II. Performance metrics for parallel systems.

Metrics	**Definitions**	**Notations**
Execution Time	Time elapsed between the start and the end	Parallel : T_p Sequential : T_s
Speedup	Performance gain between the parallel and sequential implementation	$\Psi = \frac{T_s}{T_p}$
Efficiency	Ratio of speedup to the number of processing elements	$\varepsilon = \frac{\Psi}{p}$
Cost	Sum of the time that each processing element spends	$C = T_p \cdot p$

For a parallel algorithm, the relevant operations can be put into three categories: Computations that must be performed sequentially; computations that can be performed in parallel; parallel overhead (communication operations and redundant computations). Let $\Psi(n, p)$ and $\varepsilon(n, p)$ denote the speedup and the efficiency achieved in solving a problem of size n on p processors, $\sigma(n)$ denote the inherently sequential portion of the computation, $\varphi(n)$ denote the portion of the computation that can be executed in parallel, and $\kappa(n, p)$ denote the time required for parallel overhead. Then the expressions for speedup and efficiency are

$$\Psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}, \tag{2.1}$$

$$\varepsilon(n, p) \leq \frac{\sigma(n) + \varphi(n)}{p\sigma(n) + \varphi(n) + p\kappa(n, p)}. \tag{2.2}$$

In order to analyze the performance exhibited by a parallel program, four different performance prediction formulas are demonstrated as follows [54] :

(1) *Amdahl's Law:* If f denotes the fraction of the computation that must be performed sequentially in a serial program, where $0 \leq f \leq 1$. The maximum speedup Ψ achievable by a parallel implementation with p processors performing the computation is

$$\Psi \leq \frac{1}{f + (1 - f)/p}. \tag{2.3}$$

Amdahl's law [55] provides a formula to define the relation between the speedup expectation of parallel implementation over the serial implementation when the part of the computation for the problem can be parallelized. In this law, the problem size itself is fixed when it is parallelized. And the parallel overhead cost $\kappa(n,p)$ is omitted.

(2) *Gustafson's Law:* Given a parallel program solving a problem of size n using p processors, let s denote the fraction of total execution time of the parallel program spent in serial part. The maximum speedup Ψ is

$$\Psi \leq p + (1-p)s. \tag{2.4}$$

Gustafson's Law (also known as Gustafson-Barsis' Law) [56] also suggests an upper-boundary for a parallel program to be sped up compared with a serial program. But different from Amdahl's law, Gustafson's Law has no assumption that the problem size or computation load is fixed. Therefore, it has been widely refereed to as 'scaled speedup measure'. It also deserves mentioning that the parallel overhead cost $\kappa(n,p)$ is omitted as well.

(3) *Karp-Flatt Metric:* Given a parallel computation exhibiting speedup Ψ on p processors, where $p > 1$. As the previous definition, let $\sigma(n)$ denote the inherently sequential portion of the computation, $\varphi(n)$ denote the portion of the computation that can be executed in parallel, and $\kappa(n,p)$ denote the time required for parallel overhead. The experimentally determined serial fraction e is defined to be $(\sigma(n) + \kappa(n,p))/(\sigma(n) + \varphi(n))$. Then e can be calculated as

$$e = \frac{1/\Psi - 1/p}{1 - 1/p}. \tag{2.5}$$

Because both Amdahl's law and Gustafson's Law have no consideration of parallel overhead cost $\kappa(n,p)$, the speedup may be over-estimated based on Amdahl's law and Gustafson's Law. Karp-Flatt Metric [57] defines a metric 'experimentally determined serial fraction' to reveal the aspects of the performance, such as parallel overhead, which are not easily discerned from the Amdahl's law and Gustafson's Law.

Since the metric e includes both serial time and parallel overhead cost, it can be employed to analyze the reason of the efficiency decrease. If e grows with the increase of p, then the main reason of poor parallel performance is due to the parallel overhead cost; otherwise, the inherently sequential work limits the parallel performance.

(4) *Isoefficiency Relation:* Suppose a parallel system exhibits efficiency $\varepsilon(n,p)$, where n denotes problem size, p denotes number of processors and $\kappa(n,p)$ denotes the time required for parallel overhead. Define $C = \varepsilon(n,p)/(1-\varepsilon(n,p))$. Let $T(n,1) = \sigma(n)+\varphi(n)$ denote sequential execution time, and let $T_o(n,p) = (p-1)\sigma(n)+p\kappa(n,p)$ denote the overall time cost by all processors except the time cost of the serial implementation, which includes two parts : the total time cost of the serial parts of $p-1$ processors running program and the total parallel overhead cost of p processors running program. In order to maintain the same level of efficiency with the increase of the number of processors, problem size must be increased so that the following inequality is satisfied:

$$T(n,1) \geq CT_o(n,p). \tag{2.6}$$

The isoefficiency relation [54] provides a way to evaluate the scalability of parallel methods. It can be used to determine the range of processors for which a particular level of efficiency can be maintained. Assume that the isoefficiency relation $T(n,1) \geq CT_o(n,p)$ can be represented as the explicit form of $n \geq f(p)$. Based on the isoefficiency relation $n \geq f(p)$, a parallel system is perfectly scalable if the same level of efficiency can be sustained as processors are added by increasing the size of the problem being solved. If $m = M(n)$ denotes the amount of memory required to store a problem of size n, the relation $M^{-1}(m) = n \geq f(p)$ indicates how the amount of memory used must increase as a function of p in order to maintain a constant level of efficiency. Then the scalability function $M(f(p))/p$ indicates how the amount of memory used per processor must increase as a function of p in order to maintain the same level of efficiency.

In these performance prediction formulas, Amdahl's Law can help to decide whether a program merits parallelization. Gustafson-Barsis's Law is a way to evaluate the performance of a parallel program. The Karp-Flatt metric can help to decide whether the principal barrier to speedup is the amount of inherently sequential code or parallel overhead. The isoefficiency metric is a way to evaluate the scalability of a parallel algorithm executing on a parallel computer. It can help to choose the design that will achieve higher performance when the number of processors increases.

CHAPTER III

CIRCUIT SIMULATION IN TIME AND FREQUENCY DOMAIN

A. Introduction

For general circuit problems represented by ordinary differential equations (ODEs) (or differential algebraic equations (DAEs)) in the form of $\dot{x}(t) = \psi(x(t), t)$ (or $\Psi(x, \dot{z}(x), t) = 0$), the analysis is directly related to ODE numerical solution theory (or DAE numerical solution theory). Normally, a circuit problem can be analyzed in time domain; and for certain type of circuit problems, it can be analyzed in frequency domain as well. Correspondingly, the circuit simulation techniques can be categorized into two classes : time-domain simulation methods and frequency-domain simulation methods.

In case of time-domain methods, all time derivatives are substituted by approximate expressions involving values of the quantities at discrete time instants separated by time steps that can be either fixed or variable. Then the solution of a system of ODEs is transformed in the solution of systems of algebraic equations for several time instants.

In case of frequency-domain methods for steady-state and quasi-steady state problem, the solution is assumed to have the form of a Fourier series. It can be either a classical Fourier series for periodic signals or a generalized Fourier series. The series is then substituted in the system of ODEs. Using the orthogonality of trigonometric functions with respect to an adequate inner product, a system of algebraic equations involving the Fourier coefficients of the series is obtained.

As two representative simulation techniques, time-domain transient simulation and frequency-domain harmonic balance simulation are the main focuses of our research, and correspondingly, the parallel simulation techniques in the following chapters are discussed and proposed based on these two analysis methods.

B. Time domain analysis: transient simulation

1. Introduction

Transient simulation is used for the computation of the response of an electronic circuit in the time domain. By using the modified nodal analysis (MNA) based on Kirchhoff's voltage and current laws, a circuit with n unknowns (nodal voltages and branch currents) can be formulated as

$$h(t) = \frac{d}{dt}q(x(t)) + f(x(t)) - u(t) = 0, \tag{3.1}$$

where $x(t) \in \Re^n$ denotes the vector of n unknowns, $q(x(t)) \in \Re^n$ represents the vector of the charges/fluxes contributed by dynamic elements, $f(x(t)) \in \Re^n$ represents the vector of the currents contributed by static elements and $u(t)$ is the vector of the external input excitations.

Generally, nonlinear differential equation (3.1) can only be solved in a numerical way instead of an analytical way. For this purpose, the problem defined by a system of nonlinear differential equations are converted into that of solving a sequence of systems of nonlinear algebraic equations through numerically integrating the differential equation. To do this, the time-derivative operators in the nonlinear differential equations are replaced by a finite-difference approximation, and the resulting finite-difference equations are solved time-point by time-point by using a root-finding algorithm (such as Newton's method). The discrete-time approximation employed is referred to as the integration method. In Fig. 7, a basic flow of transient simulation is shown, where $x^{i,j}$ denotes the solution of the j-th Newton iteration at the i-th time point.

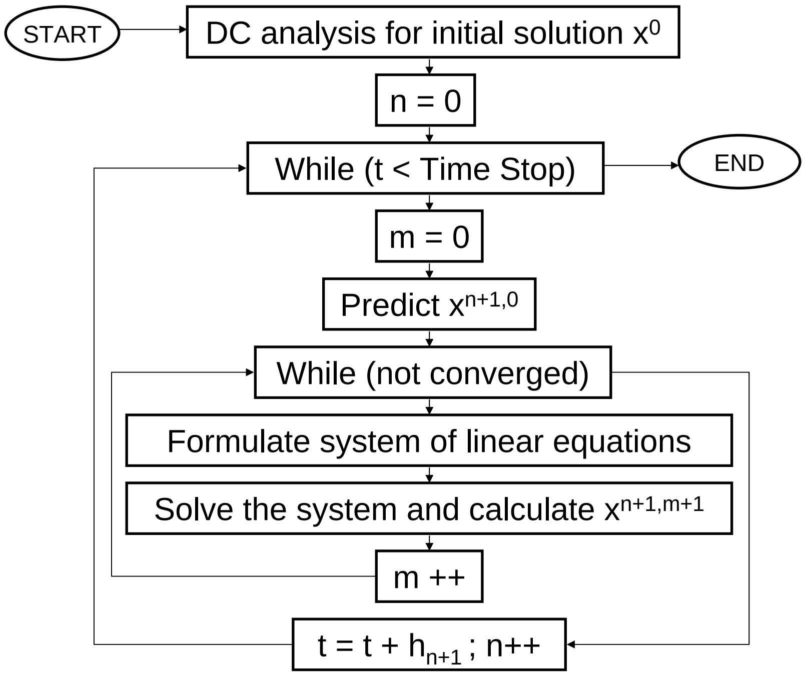
START
DC analysis for initial solution x^0
n = 0
While (t < Time Stop)
END
m = 0
Predict $x^{n+1,0}$
While (not converged)
Formulate system of linear equations
Solve the system and calculate $x^{n+1,m+1}$
m ++
$t = t + h_{n+1}$; n++

Fig. 7. A basic flow of transient simulation.

To evaluate the performance of transient simulation, two important issues deserve consideration, which are **accuracy** and **stability**. It is known that the numerical errors mainly come from numerical integration method. Therefore, in order to guarantee the accuracy of the numerical integration, the time step must be small relative to the time-constants presented in the signals. As a result, if the fixed time step is employed, the time step should be small everywhere to assure accuracy, which may lead to the inefficiency of the simulation. To address this problem, the variable time step is adopted based on certain step control mechanism. Different from accuracy, the main concern of which is the local property, Stability is a global feature of transient simulation. Since the solution at every time point is built from the solution at the previous time point, therefore the numerical error of one time point can be accumulated (or dissipated) at future time points. If the total error in the future time point does not get amplified but actually decreases with time, then the integration method is numerically stable. Since the issues of accuracy and stability are all directly relevant to the numerical integration method, some backgrounds about numerical integration methods in circuit simulation are discussed in the next section.

2. Numerical integration methods

It is known that there exist many different numerical integration methods for ODE problems [58]. However, considering issues such as accuracy, efficiency, stability and implementation, not all of them are suitable to circuit simulation. There are four types of integration methods commonly used in circuit simulation [59], which are forward Euler, backward Euler, trapezoidal rule and the backward difference formulas (also known as Gear's methods). Among these integration methods, forward and backward Euler are first order methods, meaning that the discrete-time approximation to the time-derivative operation is derived by assuming the solution trajectory is a first order polynomial over one time step. Trapezoidal rule is a second-order method, meaning that its approximation is derived by assuming that the solution trajectory is quadratic over each time step. Gear's methods are a family of methods that can be of any order. However, only the first six orders are normally used in the circuit simulator such as SPICE. It deserves mentioning that numerical integration methods can also be categorized into explicit integration method and implicit integration method. Implicit integration method like trapezoidal, backward Euler and Gear's approximations are accurate and stable, but they are computationally expensive. Explicit integration techniques like forward Euler approximation can be efficient, but they have stability problems.

Assuming fixed time steps with the time step $h = t_{k+1} - t_k$, the formulas of forward Euler, backward Euler, trapezoidal rule and 2^{nd} order Gear's method (Gear-2) can be represented as follows

Forward Euler : $\frac{d}{dt}x(t_k) \approx \frac{x(t_{k+1})-x(t_k)}{h}$,

Backward Euler : $\frac{d}{dt}x(t_{k+1}) \approx \frac{x(t_{k+1})-x(t_k)}{h}$,

Trapezoidal Rule : $\frac{d}{dt}x(t_{k+1}) \approx 2\frac{x(t_{k+1})-x(t_k)}{h} - \frac{d}{dt}x(t_k)$,

Gear-2 method : $\frac{d}{dt}x(t_{k+1}) \approx \frac{3}{2h}x(t_{k+1}) - \frac{2}{h}x(t_k) + \frac{1}{2h}x(t_{k-1})$.

In Table III, the characteristics for these integration methods are listed.

Table III. Characteristics of the numerical integration methods.

Method	Forward Euler	Backward Euler	Trapezoidal	Gear's
Step Dependency	One	One	One	Multiple
Order	First	First	Second	High
Explicit/Implicit	Explicit	Implicit	Implicit	Implicit
Stability	Partially Stable	Stable	Stable	Partially Stable

Although the orders of the numerical integration methods are different, all of them make the numerical algorithms suffer from the numerical errors. Truncation error defines the error made by replacing the time derivatives with a discrete-time approximation in numerical integration method. It is useful to consider separately the error made on each time step by using a finite-difference approximation, and the accumulated effect of the error made on each step. Correspondingly the definition of local truncation error (LTE) is the truncation error made on a single step assuming all previous steps are accurate, while global truncation error (GTE) is the maximum accumulated truncation error. Because the GTE can be treated as the collective effect of the LTEs at all discrete time points, it is critical to control the LTE to guarantee the accuracy of the numerical integration methods. The GTE is related to not only the LTE made on each step and but also the tendency of a circuit to accumulate or dissipate errors. And the latter is dependent on the stability property of the integration method.

The concept of LTE can be used to control time step in transient simulation. When using LTE-based time-step control, the LTE made on every capacitor and inductor is estimated and the time step is chosen to be small enough to assure that the largest LTE remains within tolerances. To do so, the simulator needs a measure of the LTE. Recall that N^{th} order integration methods accurately compute solutions if the trajectory follows a polynomial with an order N or less. Thus, the errors of N^{th} order methods can be approximated by $N+1^{th}$ derivatives. The simulator computes the $N+1^{th}$ derivatives and uses them as an estimation of the LTE.

C. Frequency domain analysis: harmonic balance simulation

HB simulation is a steady-state analysis technique in frequency domain, which directly analyzes the steady-state solution, avoiding the transient. In HB method, the equations are solved in the frequency domain. The key idea is the application of KCL at each node, assuming a nodal formulation is used. The frequency spectrum of all the currents at a node is balanced, i.e., KCL is applied for each independent frequency. The HB method is formulated by expressing the circuit differential equations in terms of the Fourier coefficients, and by replacing differentiation in the time domain by algebraic multiplication in the frequency domain. Each circuit variable requires many Fourier coefficients, hence the size of this system is much larger than that of the circuit differential equation. The system is typically solved using a Newton method. Compared with time-domain method for computing a circuit's steady-state solution, HB method can often be very accurate to represent the steady-state solution with a few terms of Fourier series if the steady-state is nearly sinusoidal, which is common for many analog circuits. Another advantage of HB analysis is that it is efficient to solve the quasi-periodic steady-state solution with very widely spaced fundamental frequencies.

Consider a circuit with n unknowns as shown in equation (3.1) and let the circuit be driven by a single-tone periodic excitation input source with period T. For the case of the quasi-periodic multi-tone signals, where several basic frequencies are incommensurate with each other, it can be treated by using the harmonics of a dummy fundamental frequency to represent these frequencies, then the problem can formally be solved as a single-tone case [60]. Finding the periodic steady-state solution of this circuit consists of computing the n steady-state waveforms $x(t)$ on the solution domain $t \in [0, T]$. In frequency domain, when the double-sided FFT/IFFT are used and k is the number of positive frequencies being considered, the solution waveforms $x(t)$ can be approximated as weighted finite sums of Fourier basis functions as

$$x(t) = \sum_{m=-k}^{k} X_m e^{j2\pi mt/T}, \tag{3.2}$$

which automatically satisfies the boundary conditions

$$x(t+T) = \sum_{m=-k}^{k} X_m e^{j2\pi m(t+T)/T} = \sum_{m=-k}^{k} X_m e^{j2\pi mt/T} = x(t). \tag{3.3}$$

The HB method solves for the Fourier coefficients X_m to obtain the periodic steady-state solution $x(t)$. For this purpose, the approximation (3.2), in conjunction with the circuit equations (3.1), results in the residual function:

$$h(x,t) = \sum_{m=-k}^{k} j2\pi mfQ_m e^{j2\pi mft} + \sum_{m=-k}^{k} F_m e^{j2\pi mft} - u(t), \tag{3.4}$$

where $f = 1/T$; Q_m and F_m are the Fourier coefficients of $q(x(t))$ and $f(x(t))$ with $x(t)$ the truncated Fourier series approximation of the solution waveforms. The residual function (3.4) is to be minimized on the solution domain [0, T]. This minimization is typically carried out by enforcing $h(x, t_m) = 0$ on a uniform grid of points $t_m \in$

$\{t_1, t_2, \cdots, t_N\}$ where $t_m = \frac{(m-1)T}{N}, N = 2k+1$.

As a result, the HB system of the equations corresponding to (3.1) can be formulated based on the N-point FFT and IFFT as

$$H(X) = \Omega\Gamma q(\cdot)\Gamma^{-1}X + \Gamma f(\cdot)\Gamma^{-1}X - U = 0, \tag{3.5}$$

where X is the Fourier coefficient vector of circuit unknowns $x(t)$; Ω is a diagonal matrix representing the frequency domain differentiation operator; Γ and Γ^{-1} are the N-point FFT and IFFT matrices; $q(\cdot)$ and $f(\cdot)$ are the time-domain charges/fluxes and resistive equations; and U is the input excitation $u(t)$ in frequency domain. It deserves mentioning that the nonlinear circuit devices are evaluated in the time-domain. As it can be seen in (3.5), the spectrum X is transformed into the time-domain, the time-domain response of the nonlinear device function $q(x(t))$ and $f(x(t))$ is calculated, and these waveforms are then converted back into the frequency domain.

To numerically solve the nonlinear equations in (3.5), Newton's method can be applied to solve a set of nonlinear equations of the form $H(X) = 0$ for X by starting with an initial guess $X^{(0)}$. The procedure repeatedly updates the solution by solving the linearized equation $J(X^{(k)})(X^{(k+1)} - X^{(k)}) = -H(X^{(k)})$ for $X^{(k+1)}$ until some convergence criteria are met. $J(X) = \partial H(X)/\partial X$ is called the Jacobian of H at X. At each Newton iteration of the HB problem, the Jacobian matrix can be written as [60, 61]

$$J = \Omega\Gamma C\Gamma^{-1} + \Gamma G\Gamma^{-1}, \tag{3.6}$$

where $C = diag\{c_k = \frac{\partial q}{\partial x}|_{x=x(t_k)}\}$ and $G = diag\{g_k = \frac{\partial f}{\partial x}|_{x=x(t_k)}\}$ are block-diagonal matrices with the diagonal blocks representing the linearizations of $q(\cdot)$ and $f(\cdot)$ at N sampled time points $t_1, t_2, \cdots, t_N$.

Because the explicit formulation and direct factorization of the block-dense harmonic balance Jacobian J are computationally very expensive, the direct solve of the linearized equation $JX = 0$ is not desirable. By adopting a Krylov subspace iterative method, such as Generalized Minimal Residual (GMRES) method or its flexible variant (FGMRES) [62, 63], the linearized problem defined by the Jacobian matrix may be efficiently solved. In (3.7), the algorithm of FGMRES is shown.

From the algorithm of FGMRES, it can be seen that the convergence of FGMRES depends on a good preconditioner. As shown in Fig. 8, the widely-used BD preconditioning technique discards the off-diagonal blocks in the Jacobian matrix by averaging the circuit linearizations at all discretized time points and uses the resulting block-diagonal approximation as a preconditioner [61]. For a large class of mildly nonlinear circuits, this BD preconditioner is quite effective due to the fact that the Jacobian matrix is diagonally dominant. But, this approach deteriorates for strongly nonlinear circuits where off-diagonal blocks in the Jacobian become important. To address this limitation, more efficient and robust preconditioning techniques are required to improve the performance of FGMRES solver. For this purpose, a hierarchical preconditioning technique has been proposed to take the off-diagonal blocks of the Jacobian into account. It improves the efficiency and robustness of the HB analysis, especially for strongly nonlinear circuits [64, 65]. For example in Fig. 8, instead of only maintaining the 6 on-diagonal blocks as a preconditioner, the 2 larger on-diagonal blocks are taken as a preconditioner for original problem, which considering more off-block-diagonal entries. Hierarchically, each of the 2 large blocks is solved by using smaller on-diagonal blocks as the preconditioners.

These multi-level preconditioners are created in the same fashion as that of the top-level problem by recursively decomposing a large block into smaller ones until the block size is small enough for a direct solve as shown in Fig. 9.

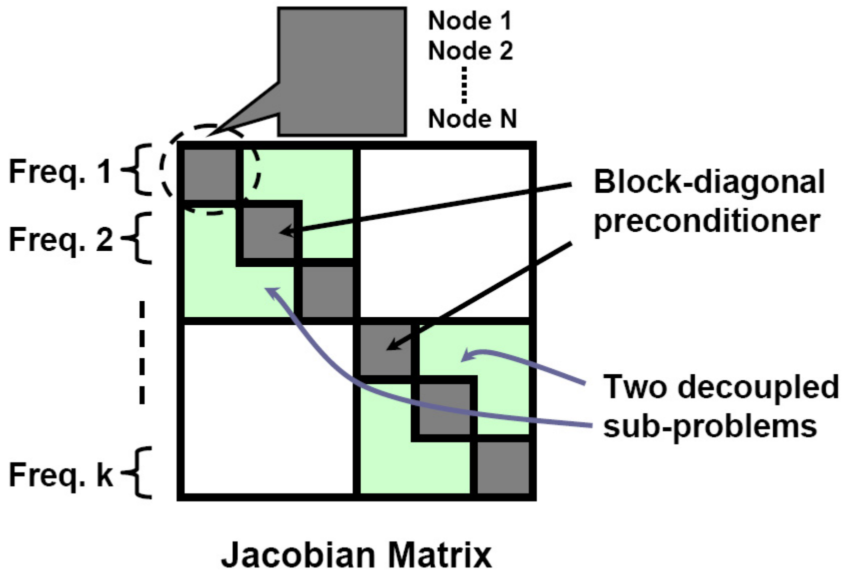

Fig. 8. BD preconditioner and hierarchical preconiditioner.

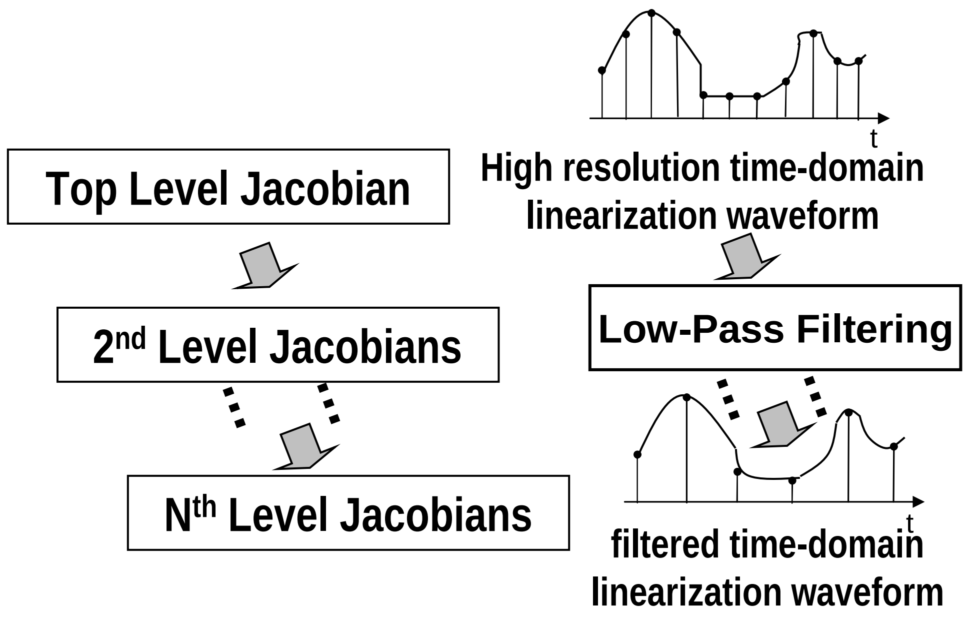
Top Level Jacobian
2nd Level Jacobians
Nth Level Jacobians
t
High resolution time-domain linearization waveform
Low-Pass Filtering
t
filtered time-domain linearization waveform

Fig. 9. Generation of hierarchial preconditioner.

```
procedure $x = FGMRES(A,\ M_i,\ b)$;
    $x_0 = M_0^{-1}b;\ r_0 = b - Ax_0;\ \beta = \|r_0\|;$
    $v_1 = r_0/\beta;\ k = 0;$
    while $(\|r_k\| > \mu(\|b\| + \|A\|\,\|x_k\|))$
        $k = k + 1;$
        $z_k = M_k^{-1}v_k; \omega = Az_{k;}$
        for $(i = 1, \cdots, k)$
            $h_{i,k} = v_i^T\omega;$
            $\omega = \omega - h_{i,k}v_i;$
        end for
        $h_{k+1,k} = \|\omega\|;$
        $v_{k+1} = \omega/h_{k+1,k};$
        $Z_k = [z_1, \cdots, z_k]; V_k = [v_1, \cdots, v_k];$
        $H_k = \{h_{i,j}\}_{1\leq i\leq j+1; 1\leq j\leq k};$
        $y_k = min\,\|\beta e_1 - H_k y\|;$
        $x_k = x_0 + Z_k y_k;\ r_k = b - Ax_k;$
    end while
end procedure
```
(3.7)

CHAPTER IV

COARSE-GRAINED WAVEPIPE PARALLEL TRANSIENT SIMULATION

A. Introduction

The wide spread of multi-core microprocessors is making parallel computing mainstream [1–3]. Unlike conventional supercomputers, modern multi-core processors are of low cost and widely accessible to typical circuit designers. With low on-chip communication overhead and high memory bandwidth as well as continuing technology scaling, multi-core or many-core systems are increasingly in a position to provide needed computing power to address many computationally intensive CAD problems. As one of the most critical forms of pre-silicon simulation and verification, SPICE-like transistor-level transient circuit analysis is indispensable to a broad range of designs including memories, custom digital and analog/RF/mixed-signal ICs [19]. The time consuming nature of transient analysis often makes it a significant design bottleneck, necessitating its parallelization.

There exist a number of parallel simulation approaches, majority of which are fine grained in nature. It is possible to parallelize the key steps in an existing simulation algorithm, e.g. device model evaluation and matrix solution. However, the efficiency of parallel matrix solvers can deteriorate fairly quickly as the number of processor cores increases. On supercomputers and computer clusters, waveform relaxation and other nonlinear relaxation methods have been proposed for parallel circuit simulation [38,39,41]. However, these methods are not widely used for robust general circuit simulation due to limited convergence properties. The efficiency of the domain decomposition approach in [42] is strongly application dependent, leading to limited applicability. Furthermore, the above two approaches require fine-grained parallel programming, hence high implementation and debugging effort.

As a coarse-grained application-level parallel approach, *WavePipe* is proposed with the recognition of two key needs in parallel application development: 1) the need to exploit domain knowledge to achieve good parallel processing efficiency by overcoming the high inter-core/thread communication overhead in fine-grained parallel approaches, and 2) the need to go beyond conventional fine-grained schemes to create a rich enough set of parallelism to fully utilize increasingly parallel computing platforms. *WavePipe* exploits application-level parallelism along the time axis by simultaneously computing the circuit solutions at multiple adjacent time points using a combination of two novel schemes: backward and forward pipelining schemes. Backward pipelining is employed in conjunction with variable step-size multi-step numerical integration methods; by moving backward along the time axis, it creates additional independent computing tasks that contribute to a larger future time step. Forward pipelining, on the other hand, facilitates predictive computing along the forward direction of the time axis.

WavePipe complements and goes beyond what can be offered by parallel device model evaluation and matrix solving and provides orthogonal opportunities for parallel computing. As a coarse grained parallel approach, *WavePipe* requires only moderate modification of existing serial simulation codes and hence is easy to implement and debug. Unlike waveform relaxation and other relaxation methods, *WavePipe* maintains the same convergency property of the standard SPICE transient analysis. Furthermore, it speeds up transient simulation without jeopardying accuracy.

B. Backward pipelining

Electronic circuits can be described by the following differential equations in time domain

$$f(x(t)) + \frac{d}{dt}q(x(t)) + u(t) = 0, \tag{4.1}$$

where $x(t)$ is the vector of nodal voltages and branch currents, $u(t)$ is the input, $f(\cdot)$ and $q(\cdot)$ are nonlinear functions describing static and dynamic nonlinearities. To solve the above nonlinear differential equations numerically, in transient analysis a numerical integration method, such as Backward Euler (BE) or Trapezoidal Rule (TR), is applied to convert (4.1) to a sequence of nonlinear algebraic equations. The concept of the local truncation error (LTE) is used to control the errors incurred in numerical integration, resulting in the LTE-based time step control [66]. The idea is to limit the time step size such that a pre-defined local truncation error tolerance is satisfied.

In standard transient analysis, time-domain circuit responses are computed sequentially along the time axis such that for the solution of any time point, the known responses of the preceding points provide a well defined initial condition. At the first glance, in both one-step and multi-step integration methods, the predetermined data dependency seemingly makes it impossible to enable parallel computing along the time axis, as illustrated in Fig. 10. However, as will be described, variable-step size

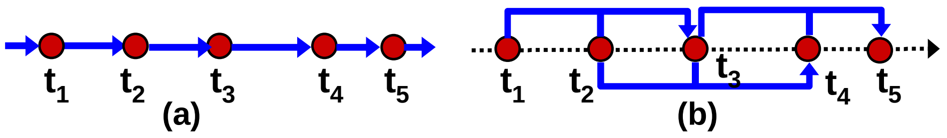

Fig. 10. Data dependency in a) one-step, and b) multi-step (2-step) numerical integration.

multi-step methods can be indeed exploited for parallel computing, but via some new perspectives.

1. Variable-step size multi-step methods

The multi-step Gear's integration formulae have the following form [67]

$$x_{n+1} = \beta_0 \dot{x}_{n+1} + \sum_{k=1}^{p} \alpha_k x_{n+1-k}, \tag{4.2}$$

where p is the order of numerical integration, x_i, $i = n+1-p, \ldots, n+1$ is the circuit response at time point i, x_{n+1} is the unknown circuit response at time point $(n+1)$, and (β_0, α_k) are certain coefficients, which are constant in fixed-step Gear's methods. Consider the special case of the two-step variable time-step Gear's method [68]

$$\begin{aligned} x_{n+1} \quad = \quad & -x_{n-1}\frac{h_{n+1}^2}{h_n(2h_{n+1}+h_n)} + x_n\frac{(h_{n+1}+h_n)^2}{h_n(2h_{n+1}+h_n)} \\ & +\dot{x}_{n+1}\frac{h_{n+1}(h_{n+1}+h_n)}{2h_{n+1}+h_n}, \end{aligned} \tag{4.3}$$

where $h_{n+1} = t_{n+1} - t_n$, $h_n = t_n - t_{n-1}$. The local truncation error of (4.3) is

$$\varepsilon_{n+1} = -\frac{h_{n+1}^2(h_{n+1}+h_n)^2}{6\cdot(2h_{n+1}+h_n)} \cdot x^{(3)}(\tau), \tag{4.4}$$

where τ is in $[t_n,\ t_{n+1}]$, and the third order derivative $x^{(3)}(\tau)$ may be approximated as $x^{(3)} \approx 3!DD_3(t_{n+1},\ t_n,\ t_{n-1},\ t_{n-2})$, in which $DD_3(t_{n+1},\ t_n,\ t_{n-1},\ t_{n-2})$ denotes the third order divided difference evaluated at time points t_{n+1}, t_n, t_{n-1}, t_{n-2}. As a standard Gear2 method, this integration formula is stiffly stable [68].

The above formulae can be extended to a three-step one

$$x_{n+1} = \beta_0 \dot{x}_{n+1} + \alpha_1 x_n + \alpha_2 x_{n-1} + \alpha_3 x_{n-2}. \tag{4.5}$$

Define a set of new variables: $T_1 = t_{n+1} - t_n$, $T_2 = t_{n+1} - t_{n-1}$, $T_3 = t_{n+1} - t_{n-2}$. The coefficients in (4.5) can be obtained by solving

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ T_1 & T_2 & T_3 & -1 \\ T_1^2 & T_2^2 & T_3^2 & 0 \\ T_1^3 & T_2^3 & T_3^3 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{4.6}$$

which leads to

$$\begin{aligned} \alpha_1 &= \tfrac{T_2^2 T_3^2}{(T_2-T_1)(T_3-T_1)(T_1T_2+T_1T_3+T_2T_3)} \\ \alpha_2 &= \tfrac{T_1^2 T_3^2}{(T_1-T_2)(T_3-T_2)(T_1T_2+T_1T_3+T_2T_3)} \\ \alpha_3 &= \tfrac{T_1^2 T_2^2}{(T_1-T_3)(T_2-T_3)(T_1T_2+T_1T_3+T_2T_3)} \\ \beta_0 &= \tfrac{T_1T_2T_3}{T_1T_2+T_1T_3+T_2T_3} \end{aligned} . \tag{4.7}$$

The local truncation error of (4.5) is given by

$$\varepsilon_{n+1} = \frac{T_1^2 T_2^2 T_3^2}{24(T_1T_2 + T_2T_3 + T_1T_3)} \cdot x^{(4)}(\tau), \tag{4.8}$$

where τ is in $[t_n,\, t_{n+1}]$ and $x^{(4)}(\tau)$ is approximated by $x^{(4)} \approx 3!DD_4(t_{n+1},\, t_n,\, t_{n-1},\, t_{n-2},\, t_{n-3})$, in which $DD_4(t_{n+1},\, t_n,\, t_{n-1},\, t_{n-2},\, t_{n-3})$ denotes the fourth order divided difference evaluated at time points $t_{n+1},\, t_n,\, t_{n-1},\, t_{n-2},\, t_{n-3}$.

In the LTE-based time-step control, the largest time step size that does not exceed a specified local truncation error tolerance is chosen. This strategy ensures that the numerical integration error incurred at each time step is well controlled while the transient simulation can be advanced in time as fast as possible. In the two-step Gear's method presented above, the time step can be estimated based on LTE as follows. For given $h_n = t_n - t_{n-1}$, $DD3$ and a specified LTE tolerance ε, let the time step to be determined as $h_{n+1} = kh_n$, $k > 0$. According to (4.4), the maximum allowable h_{n+1} can be determined by solving the following equation

$$\frac{k^2(k+1)^2}{(2k+1)} = \left|\frac{\varepsilon}{DD_3 \cdot h_n^3}\right|. \tag{4.9}$$

In addition to LTE, other factors may be further considered when choosing a suitable time step. Because nonuniform step-sizes modify the stability region of the integration method, time step may not be changed too rapidly in order to assure that the stability property does not depart considerably from that of a uniform step-size method. For instance, step-size variations can be constrained such that $\frac{1}{\alpha} \leq k \leq \alpha$, where $\alpha > 0$ and is not too large.

2. Backward pipelining

Without loss of generality, backward pipelining is proposed under the context of double-threaded two-step variable time-step integration methods. The discussion can be easily extended to other multi-threaded scenarios with a higher-order integration method. To simplify the discussion, assume that the third order divided difference $DD3$ remains constant at different time points although in practice variable $DD3$ can be easily handled. Under this assumption, the LTE is only a function of the two time steps, h_n and h_{n+1}, in (4.4).

Let us first consider a naïve approach as shown in Fig. 11 (a). The circuit responses at three time points $T_1 - T_3$ are assumed to be known. Using the solutions at T_2 and T_3 as the initial conditions, a thread may be launched to compute the solution at T_5. One may attempt to use a second thread to compute the solution at T_4 by using solutions at T_1 and T_2 as the initial conditions. This choice may seemingly make use of two processing elements (threads) to allow for parallel computing. However, a more careful look reveals that the work done by the second thread at T_4 is almost always useless. This is because T_5 is usually beyond T_4 due to the use of the most

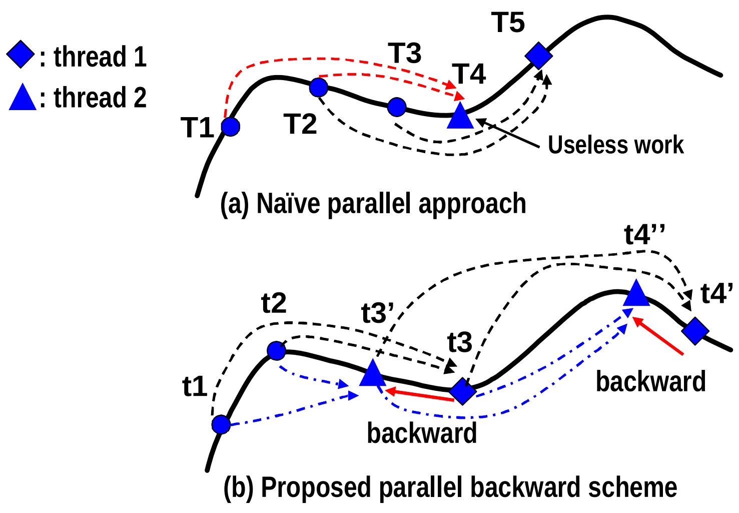

Fig. 11. Parallel double-threaded backward pipelining: a) an naïve approach, and b) the proposed backward pipelining.

updated initial conditions. The solution at T_4 only provides an interpolation point between T_3 and T_5 and by itself does not contribute to a faster transient analysis.

To exploit the variable-step size two-step methods (e.g. Gear2) in a more meaningful way, parallel backward pipelining is proposed as shown in Fig. 11 (b). While the first thread is solving the transient circuit response at a time point that is determined by the standard numerical integration and LTE step control, a second thread is launched to solve the solution at a preceding time point in parallel, but based on the same latest available initial conditions. To see how this can speed up the transient analysis, let us examine the dependency of the LTE of the two-step numerical integration on the two time steps, h_n and h_{n+1}, in (4.4). The partial derivatives of the LTE with respect to h_n and h_{n+1} can be shown to be

$$\begin{aligned}\frac{\partial\left|\varepsilon\right|}{\partial h_{n+1}} &= \frac{2\left|DD_3\right|\begin{bmatrix} h_{n+1}(h_{n+1}+h_n)\cdot \\ \left(h_{n+1}^2+(h_{n+1}+h_n)(2h_{n+1}+h_n)\right)\end{bmatrix}}{(2h_{n+1}+h_n)^2} \\ \frac{\partial\left|\varepsilon\right|}{\partial h_n} &= \frac{\left|DD_3\right| h_{n+1}^2(h_{n+1}+h_n)(3h_{n+1}+h_n)}{(2h_{n+1}+h_n)^2}\end{aligned} \tag{4.10}$$

Both derivatives are positive, which implies that the LTE increases with both h_n and h_{n+1}, as they are expected. Also, in order to explain the reason why the solution at the second thread (backward thread) is helpful to push the future time point further, it requires to prove that the smaller is the time step h_n, the larger the time step h_{n+1}. An analysis has been shown as follows:

$$\begin{aligned}&\varepsilon_{n+1} = -\tfrac{h_{n+1}^2(h_{n+1}+h_n)^2}{6\cdot(2h_{n+1}+h_n)}x^{(3)}(\tau) \\ &\Rightarrow C \equiv -\tfrac{6\varepsilon_{n+1}}{x^{(3)}(\tau)} = \tfrac{h_{n+1}^2(h_{n+1}+h_n)^2}{(2h_{n+1}+h_n)} \\ &\Rightarrow h_{n+1}^2\left(h_{n+1}+h_n\right)^2 - C\left(2h_{n+1}+h_n\right) = 0 \\ &\Rightarrow 2h_{n+1}\left(h_{n+1}+h_n\right)^2\tfrac{\partial h_{n+1}}{\partial h_n} + 2h_{n+1}^2\left(h_{n+1}+h_n\right)\left(\tfrac{\partial h_{n+1}}{\partial h_n}+1\right) - C\left(2\tfrac{\partial h_{n+1}}{\partial h_n}+1\right) = 0 \\ &\Rightarrow \tfrac{\partial h_{n+1}}{\partial h_n} = \tfrac{C-2h_{n+1}^2(h_{n+1}+h_n)}{2(h_{n+1}(h_{n+1}+h_n)(2h_{n+1}+h_n)-C)} \\ &\Rightarrow \tfrac{\partial h_{n+1}}{\partial h_n} = \tfrac{\frac{h_{n+1}^2(h_{n+1}+h_n)^2}{(2h_{n+1}+h_n)}-2h_{n+1}^2(h_{n+1}+h_n)}{2\left(h_{n+1}(h_{n+1}+h_n)(2h_{n+1}+h_n)-\frac{h_{n+1}^2(h_{n+1}+h_n)^2}{(2h_{n+1}+h_n)}\right)} \\ &\Rightarrow \tfrac{\partial h_{n+1}}{\partial h_n} = \tfrac{-h_{n+1}^2(h_{n+1}+h_n)(3h_{n+1}+h_n)}{2h_{n+1}(h_{n+1}+h_n)\left(3h_{n+1}^2+h_n^2+3h_{n+1}h_n\right)} < 0\end{aligned} \tag{4.11}$$

Since the derivative is negative, it is true that the time step h_{n+1} increases when the time step h_n decreases.

In Fig. 11(b), it is assumed that the transient responses at t_1 and t_2 are already computed. As in a standard two-step integration method with LTE based time step control (e.g. (4.3, 4.9)), the first thread is launched to compute the circuit solution at t_3 using the latest initial conditions at t_1 and t_2. In parallel, a second thread is started to solve the solution at t'_3, which is in between t_2 and t_3, using the same initial conditions. The computation of the t'_3 circuit response is referred as a backward step. Importantly, it shall be noted that since the LTE tolerance is satisfied at t_3, so is it at any placement of t'_3 that is between t_2 and t_3. Because of this, no accuracy issue is incurred. The work done at t'_3 can be used in a meaningful way as follows. Upon the completion of the both threads, the solutions at t_3 and t'_3 will be used as the initial conditions for future time points. Compared with the serial transient simulation where the solutions at t_2 and t_3 are used as the initial conditions for the next time point, the availability of the t'_3 solution in parallel backward pipelining reduces the value of h_n in the LTE based step control as in (4.9). Hence, backward pipelining leads to a larger next time step and advances the transient analysis faster along the time axis. The same two-thread pattern is repeated for solving the solutions at future time points such as t_4 (t'_4).

In practice, the location of t'_3 (or t'_4) shall be chosen to balance between efficiency and numerical stability. From an LTE point of view, placing t'_3 closer to t_3 allows for a larger next time step while making them too close to each other may introduce numerical problems. In the implementation, a damping factor γ is used to determine the location of t'_3 such that $t'_3 - t_2 = \gamma \cdot (t_3 - t_2)$, $0 < \gamma < 1$. γ can be tuned experimentally to optimize the runtime while maintaining good numerical robustness. The double-threaded backward pipelining can be generalized for multi-step methods. A three-thread parallel scheme has been developed where two threads are launched to simultaneously compute the circuit solutions at two backward steps, which contribute to an improved future time step size in the three-step Gear's method.

C. Forward pipelining

The proposed parallel backward pipelining helps advance the transient analysis by providing better initial conditions in conjunction with multi-step integration methods. A forward scheme is also proposed, which directly computes one or multiple future time points in parallel. Consider the situation shown in Fig. 12. It is assumed that the transient solutions at time points t_1 and t_2 are already computed. Using an LTE-based variable step-size two-step integration method, one thread may be used to compute the circuit solution at t_3 using the solutions at t_1 and t_2 as the initial conditions. Again, as an nave approach, one may attempt to use a second thread to compute the response at a time point t_4 that is further down using the same initial conditions. While this seems to allow for two independent computation tasks running in parallel, the solution obtained at t_4 may not be trusted if the maximum permissible time step is already employed at t_3. In other words, the LTE tolerance may not be satisfied at t_4.

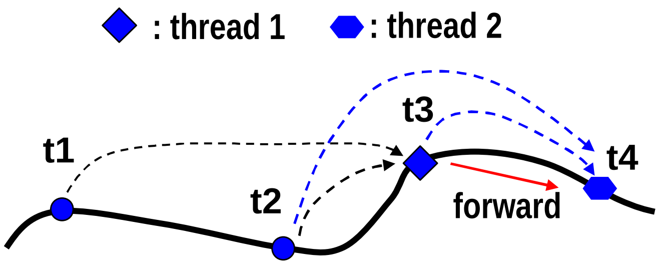

Fig. 12. Double-threaded forward pipelining.

This problem is remedied in the proposed forward scheme, which is shown in Fig. 12. While the first thread is working at t_3 using the t_1 and t_2 solutions as the initial conditions, a second thread is started to compute the solution at t_4. Here, the key difference is that the t_4 solution is based on using the solutions at t_2 and t_3 as the initial conditions. Obviously, this creates data dependency between the two threads since the solution at t_3 that is being computed by the first thread is not available yet. The data dependency and resulting issues are resolved as follows.

1. Prediction of time step size

To start the work at t_4 in parallel, the time step $h_f = t_4 - t_3$ must be decided first. h_f also depends on the unknown circuit solution at t_3, say $x(t_3)$. To address this difficulty, an estimate of $x(t_3)$ is quickly computed and used to launch the second thread. In particular, Forward Euler (FE) rule is employed to get the estimate, say $\tilde{x}(t_3)$. Since FE is explicit, such estimation can be done very efficiently. However, several complications arise and must be addressed. Using $\tilde{x}(t_3)$ may lead to an overly optimistic time step size h_f. If this happens, the resulting solution at t_4 does not satisfy the LTE tolerance and must be discarded. To reduce the chance of time step overestimation, a damping factor β ($\beta < 1.0$) is introduced to scale down the estimated time step such that a more conservative time step is used: $h_{f,damped} = \beta h_{f,FE}$. If the LTE is still violated even with the use of damping upon the availability of the exact t_3 solution, the predictive work done at t_4 will have to be revoked, as detailed later in the dissertation.

2. Accuracy and stability

Apart from the possibility of step size overestimation, the circuit solution computed in parallel at t_4 using $\tilde{x}(t_3)$ may not be accurate even if the time step is estimated conservatively. In this regard, the accuracy is guaranteed by one of the two inter-thread communication approaches in Fig. 13. As shown in Fig. 13 (a), in the coarse-grained approach at least one nonlinear Newton iteration is performed at t_4 after the solution at t_3 has fully converged. This guarantees that the t_4 solution computed by thread 2 will converge to the exact value based upon the converged solution at t_3. It also implies one inter-thread communication in which thread 2 loads the converged $x(t_3)$ computed by thread 1.

In the fine inter-thread communication, thread 2 more frequently updates $\tilde{x}(t_3)$ as the convergence progress is being made by thread 1, as shown in Fig. 13 (b). The updated $\tilde{x}(t_3)$ that is available at the end of each Newton iteration in thread 1 may be subsequently accessed to start the following Newton iteration in thread 2. Like before, upon the completion of thread 1, the fully converged $x(t_3)$ is loaded by thread 2 as the part of exact initial conditions to perform one or more Newton iterations to guarantee the accuracy of the t_4 solution. Here, since the initial conditions are more frequently updated, thread 2 is made to converge faster, however, at the cost of somewhat higher inter-thread communication overhead. In the experiments, it is observed that the use of the finer grained communication scheme indeed further speeds up the transient simulation of large circuits, where the cost of Newton iterations is more dominant. Furthermore, it shall be noted that Forward Euler is only employed to estimate initial conditions and the simulation accuracy is strictly guaranteed at any time point using a stable integration method. Hence, the use of the FE based estimation does not incur any stability concern.

D. Multi-threaded WavePipe and thread scheduling

The backward and forward schemes can be combined to create a variety of multi-threaded WavePipe implementations to utilize a larger number of processor cores, particularly when further combined with low-level parallel schemes such as parallel device model evaluations and matrix solvers. For the purpose of discussion, low-level parallelization is not considered. For a fixed number of threads/cores, multiple par-

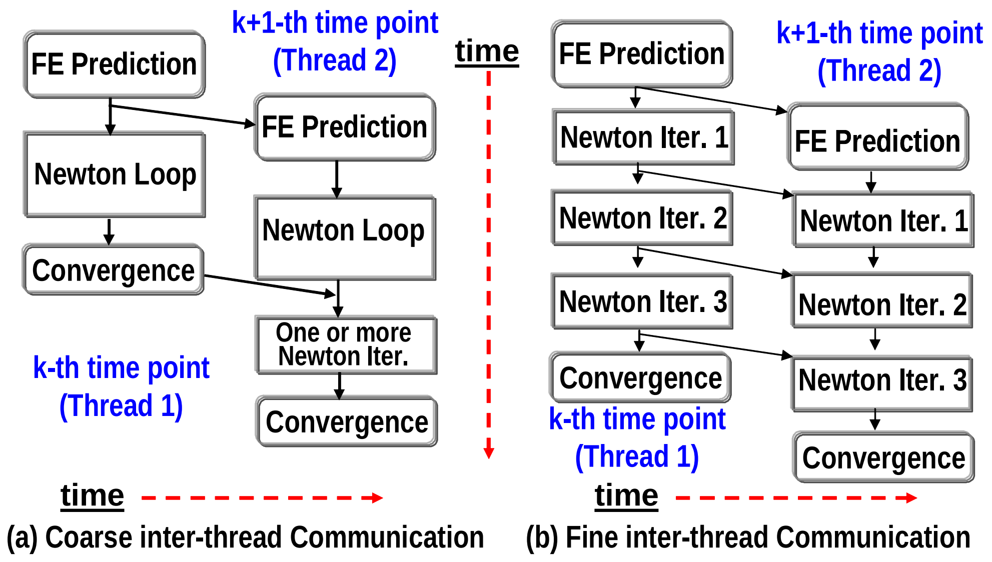

Fig. 13. Fine and coarse grained inter-thread communications to guarantee the accuracy of the parallel forward scheme.

allelizing schemes exist. For instance, a three-thread WavePipe may contain a thread that is allocated for standard variable-step size and multi-step integration (referred to as the base thread), one forward thread and one backward thread. Alternatively, the last two threads may be replaced by two forward threads. In the following, the threading scheduling issues in multi-threaded WavePipe are discussed using a four-thread (4T) implementation as an example, which contains one base thread, one backward thread, and two forward threads.

1. Thread scheduling

As shown in Fig. 14, in this 4T implementation, starting from the two known circuit solutions as the initial conditions, the base thread T1 computes the circuit solution at the next time point according to a standard numerical integration method, in this case, Gear2. T1 first determines the time step and then computes an FE based estimation of the new solution. The FE estimation is used as an initial guess for the circuit response at the new time point. It is also employed in forward pipelining to enable parallel predictive computing along the time axis, as described in Section 1. The second thread T2 is simultaneously launched to compute the circuit response according to backward pipelining. Upon the availability of the FE estimation computed by T1, the third thread T3 is started to facilitate the forward scheme. T3 also performs an FE based estimation for the circuit solution it will compute, which provides a basis to launch the second forward scheme thread, T4. The launching and completion of such four threads as a whole is referred to as a *thread scheduling cycle*. It shall be noted that within one scheduling cycle, T2 may complete slightly before T1. This is because that T2 works on a new time point that has a smaller time step than that of T1. The dependency between T1, T3 and T4 makes these three threads finish one after another.

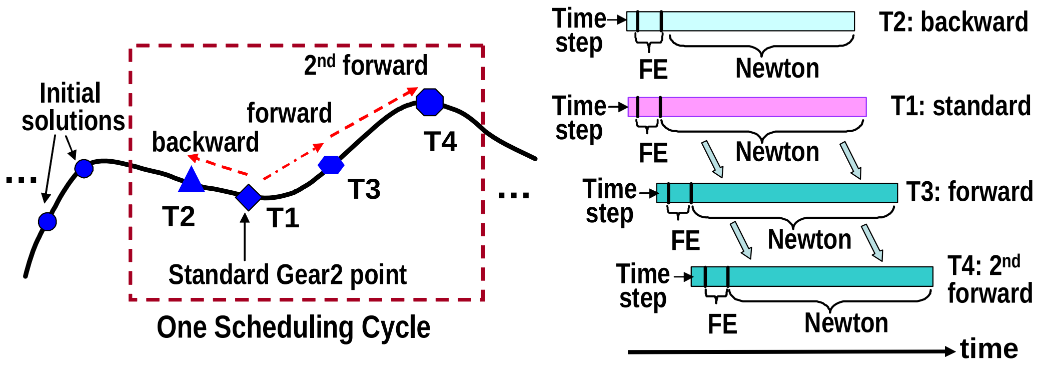

Fig. 14. Four-thread waveform pipelining.

Once a scheduling cycle is completed, another cycle starts in the same fashion. As shown in Fig. 15 (a), if all the threads successfully complete in a scheduling cycle, the next cycle will start using the solutions computed by T3 and T4 as the initial solutions. However, as described in Section 1, it is possible to overestimate the time step in forward pipelining. If this happens, the work done by the corresponding thread is discarded. In Fig. 15 (b), it is assumed that T3 overestimates its time step.

Hence, the solution computed by T3 does not satisfy the LTE tolerance and shall be discarded. Due to the data dependency between T3 and T4, the work done by T4 is discarded as well. In this case, the next scheduling cycle will use the solutions computed by T1 and T2 as the initial conditions.

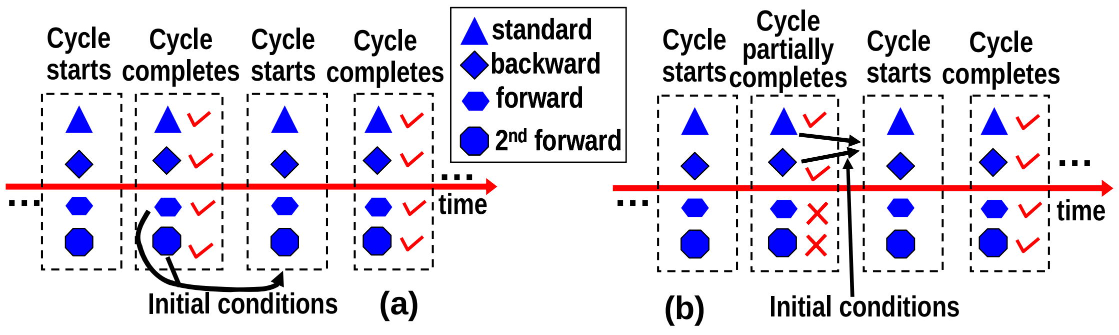

Fig. 15. 4T waveform pipelining: a) without revoking of forward pipelining and b) with revoking of forward pipelining.

2. Scheduling policies

The discussion above outlines a basic thread scheduling policy. In practice, multiple alternatives exist, which provide a basis for performance tuning of the parallel WavePipe implementation. For example, thread scheduling can be done with a finer granularity than a thread scheduling cycle. Note that within a cycle the four threads may complete at different times. By algorithm construction, forward pipelining threads complete subsequently after the base thread. It is possible to launch new work immediately using any available thread without waiting the entire scheduling cycle to finish. However, this may or may not be beneficial depending on when the remaining running threads in the cycle can finish. The damping factor β introduced in Section 1 can be varied to modify the amount of conservativeness in the time step estimation in forward pipelining. A larger β value may help advance the transient analysis more rapidly, however, with a higher risk of revoking the work of forward threads.

In the implementation, multiple thread scheduling policies are implemented. The optimal policy together with the optimal values of various control parameters are experimentally selected to optimize the overall parallel simulation efficiency.

E. Experimental results

WavePipe is implemented in C/C++ using Pthreads library on a high-end shared-memory Linux server with four dual-core processors. To verify the performance of *WavePipe* on a variety of circuits, eight test circuits with distinguishing characteristics as shown in Table IV, are used in the experiments. Since the serial SPICE-like Backward Euler's (BE) method and the serial two-step Gear's method have comparable performances, only the results of the former are shown in Table IV, which are used as a reference to evaluate various parallel schemes. In Table IV, columns labeled as *Size*, *Points* and *Runtime* are the number of circuit unknowns, number of time points simulated and total transient simulation runtime, respectively. Note that there are nonlinear drivers in the two RLC mesh circuits.

Table IV. Statistics of test circuits and serial BE.

IDX	Circuit	Size	Points	Runtime(s)
1	VCO	20	90,545	52
2	Power Amplifier	8	118,426	40
3	DB Mixer	27	140,273	65
4	Ring Oscillator	61	115,973	285
5	Frequency Divider	17	45,693	24
6	Digital Adder	112	2,619	13
7	RLC Mesh 1	13,097	680	3,032
8	RLC Mesh 2	27,670	146	2,970

As described in Section 2, the three-thread parallel backward pipelining is implemented using the three-step Gear's method. Compared with the two-thread two-step Gear based backward pipelining, up to 45% runtime speedup can be achieved. However, in practice, the stability issue has to be more carefully considered for higher

Table V. Runtime speedups of 2-threaded coarse-grained WavePipe schemes.

	2T 1-backward		2T 1-forward	
IDX	T(s)	Speedup	T(s)	Speedup
1	36	1.46	30	1.72
2	28	1.44	23	1.74
3	48	1.35	40	1.64
4	194	1.47	161	1.77
5	19	1.24	16	1.51
6	10	1.25	8	1.53
7	2,445	1.24	1969	1.54
8	2,376	1.25	1904	1.56

Table VI. Runtime speedups of 3-threaded coarse-grained WavePipe schemes.

	3T 1-backward -1-forward		3T 2-forward	
IDX	T(s)	Speedup	T(s)	Speedup
1	27	1.96	26	2.02
2	21	1.93	20	2.02
3	36	1.82	34	1.91
4	148	1.92	140	2.03
5	14	1.69	13	1.77
6	8	1.69	7	1.80
7	1827	1.66	1703	1.78
8	1727	1.72	1632	1.82

order Gear's methods. The results presented in the following are based upon the two-step Gear's method.

In Table V, Table VI and Table VII, the runtimes and speedups (w.r.t serial BE) of six coarse-grained WavePipe schemes are listed, which are 2-thread backward pipelining, 2-thread forward pipelining, 3-thread one-backward-one-forward pipelining, 3-thread two-forward pipelining, 4-thread two-forward-one-backward pipe-lining, and 4-thread three-forward pipelining, respectively. Note that in all these schemes, there exists a base thread that implements the standard numerical integration. The average runtime speedups of the six schemes are 1.33x, 1.62x, 1.80x, 1.89x, 2.14x and 2.26x, respectively. In Fig.16, the runtime speedups of four of these six WavePipe schemes are visually presented. It is observed that the runtime scales almost linearly with the number of threads. In Fig. 17, a real-time profiling of the 3-thread one-forward-one-backward scheme is shown running on an RLC mesh circuit. The

Table VII. Runtime speedups of 4-threaded coarse-grained WavePipe schemes.

IDX	4T 1-backward -2-forward		4T 3-forward	
	T(s)	Speedup	T(s)	Speedup
1	22	2.35	21	2.46
2	18	2.26	17	2.35
3	29	2.21	28	2.32
4	128	2.23	119	2.39
5	12	2.05	11	2.15
6	6	2.04	5	2.18
7	1524	1.99	1458	2.08
8	1463	2.03	1381	2.15

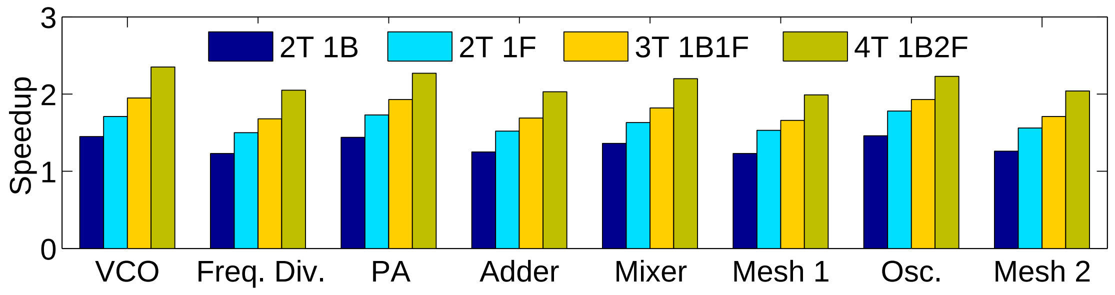

Fig. 16. Speedups of various WavePipe schemes.

runtimes of the three threads are break down to the following categories: time step computation, Forward Euler initial solution estimation, and remaining computation of one time step circuit response. The latter is further distinguished according to the mode of operation: base, forward (FWD) and and backward pipelining (BWD).

Next, the proposed coarse-grained parallel WavePipe is compared with the low-level scheme that bases upon parallel transistor device model evaluation and matrix solving. The public domain parallel matrix solver SuperLU [69] is employed. The comparison is made using a double-balanced mixer and an RLC mesh circuit in Fig. 18 and Fig. 19. When the number of threads is in between 2 and 4, the new schemes are completely based upon the proposed WavePipe. The 8-thread new scheme demonstrates the possibility of combining Wavepipe, in this case, the three-forward scheme, with parallel device model evaluation and matrix solver. In particular, within each

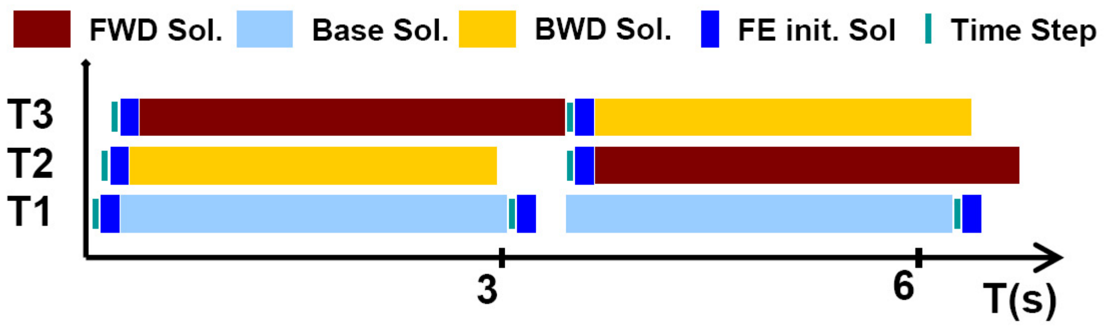

Fig. 17. Realtime thread profiling of the 3T one-forward-one-backward waveform pipelining.

mode of operation (one base mode, three forward modes), two threads are utilized to facilitate device model evaluation and matrix solving. When the number of threads varies from 2 to 4, WavePipe is comparable to the low-level parallel scheme. However, the runtime scaling of the low-level scheme already starts to saturate beyond four threads. The eight-thread parallel model evaluation/matrix solving scheme does not further improve the runtime. This clearly underscores the need to develop new application-level coarse-grained parallelizing avenues, which is the focus of this work, especially on massively parallel platforms. In contrast, the combined 8-thread scheme brings favorable speedups.

F. Summary

A coarse-grained *Waveform Pipelining* approach to parallel transient circuit simulation is proposed. The backward and forward pipelining schemes allow us to exploit parallel computing along the time axis, hence offering new avenues to utilize application-level parallelism. The experiments have demonstrated good efficiency factor of the proposed approach and its promising potential on parallel computing platforms with large numbers of processing cores.

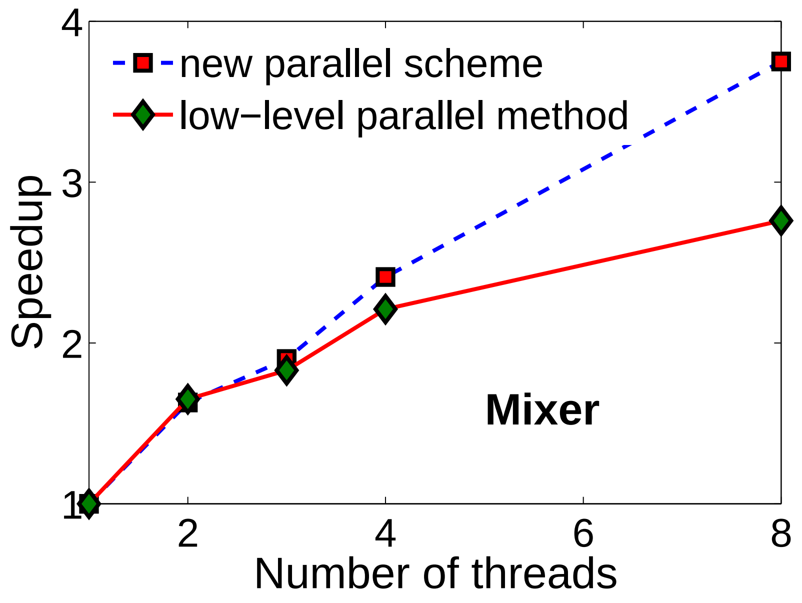

Fig. 18. Comparison between WavePipe and low-level parallel model evaluation/matrix solving: double-balanced mixer.

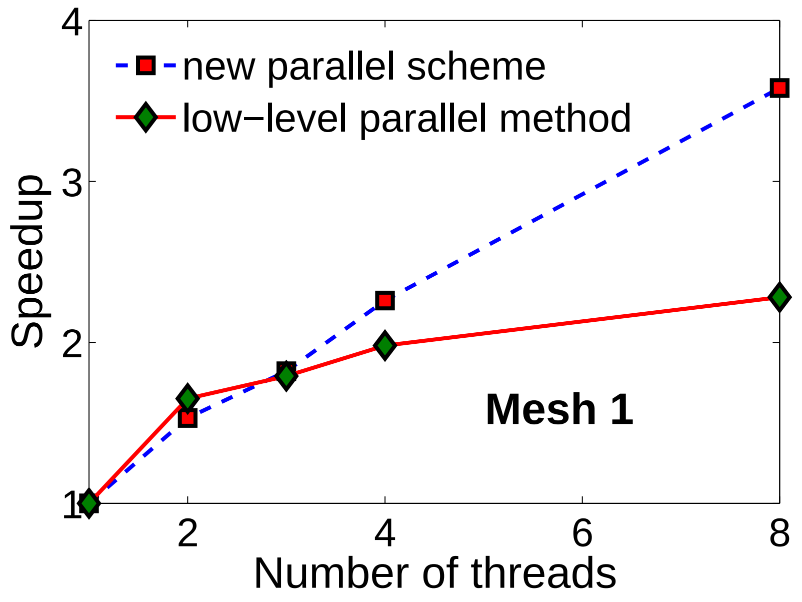

Fig. 19. Comparison between WavePipe and low-level parallel model evaluation/matrix solving: RLC mesh.

CHAPTER V

PARALLEL TRANSIENT SIMULATION BASED ON EXPLICIT INTEGRATION METHOD

A. Introduction

As demonstrated in the previous chapter,a large body of research has been devoted to improve the performance and runtime efficiency of transistor-level transient analysis via algorithmic innovation and parallelization [38, 39, 41, 42, 48, 70–72]. It deserves mentioning that the explicit integration methods have been exploited in [71, 72]. Different from the widely-used implicit integration methods, such as backward Euler (BE) and trapezoidal rule (TR), the explicit integration methods, such as forward Euler (FE), have their own benefits for circuit simulation. An explicit method (e.g. FE) efficiently extrapolates the transient response at each future time point, circumventing the need for solving any linear or nonlinear system of equations. However, this potential computational advantage comes with a severe limitation: instability. Usually, the largest time step of an explicit method is limited by the smallest time constant in a circuit, presenting a severe limitation for many practical circuits with widely separated time constants. In many cases, the stability limitation of explicit methods offsets the computational benefit obtained from extrapolation. As a result, explicit methods are not widely used in practical SPICE implementations. To address this problem, different ideas are suggested in [71, 72]. In particular, in the fast ACES simulator developed at IBM [71], the stability issue is alleviated through a number of heuristics. However, such techniques are heuristics in nature and are only applicable to fast digital timing applications.

In this chapter, efficient and stable explicit numerical integration method is proposed. This research leverages on the very recent development on numerical solution of ordinary differential equations (ODEs) from the numerical analysis community. The so-called explicit *projective* and *telescopic projective* integration methods have been shown to be efficient for certain physical simulation problems with widespread time constant distributions [73, 74]. In this dissertation, the same principles can be applied to circuit simulation so that the *effective* time step of such a multi-level explicit integration scheme is no longer limited by the smallest time constant in the circuit and can be made comparable to the largest time constant while fully guaranteeing stability. Through the prototype implementation, it can be seen that the telescopic projective integration method addresses the known stability limitation of explicit numerical integration with a theoretically sound foundation, leading to fast explicit circuit simulation. Equally important, the explicit nature of the approach breaks the entire simulation task into independent sub-tasks of device model evaluation, small-scale node (or device) based system solves, facilitating natural parallelization. The relaxation of matrix solutions, which is very difficult to parallelize with good efficiency, presents a significant advantage.

B. Principle of telescopic projective integration

The telescopic projective framework in essence is a multi-level numerical integration method for solving initial value ODE problems [73, 74]. To better understand the key idea of the telescopic projective integration method, let us start from a stiff ODE problem.

Consider a stiff ODE problem in the form: $\dot{X} = AX$, where the distribution of the time constants (eigenvalues of A) are widely spread out and there are gaps between them. As a simple example shown in Fig. 20, where there only exists one gap between the two clusters of eigenvalues (G1 and G2), corresponding to the fast and slow components in the system. Under the context of circuit simulation, it is known that fast components exist for a short period of time and dissipate quickly while the long-time circuit transient response is mainly determined by the slow components. For practical purposes, it is often sufficient to only track the slow components in the transient response. Hence, it is desirable to use time steps with a size comparable to large time constants of the system to gain simulation efficiency. Unfortunately, the use of explicit integration methods is severely limited by stability concerns; the largest time step must be set to be comparable to the smallest time constants in the system to ensure stability. It is especially inefficient where there exists a large gap between the time constants of fast and slow components. Projective integration is specifically designed to accelerate the solving of ODE problems under such a situation.

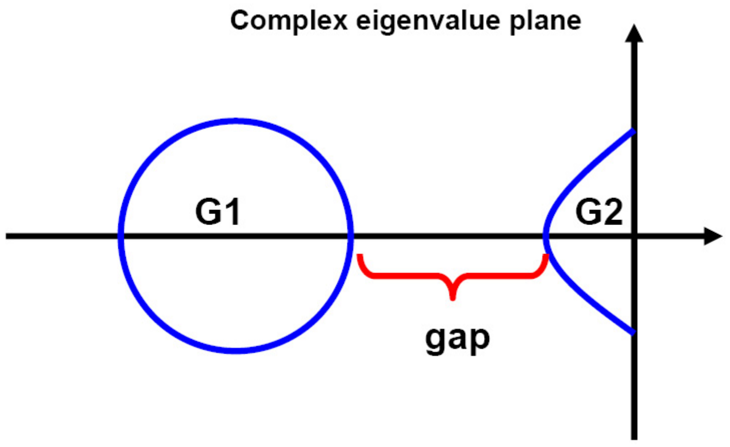

Fig. 20. Distribution of the eigenvalues with a single gap.

In projective integration, a combination of an *inner* integrator and *outer* projective integrator is employed to achieve efficiency and stability at the same time. Intuitively, to ensure the stability, a number of integration steps with a relatively small time step are taken corresponding to the fastest time constant at the 'inner' loop to heavily damp the fast components in the system. Therefore, the accumulated integration error is exponentially damped. In other words, the purposely chosen small time step in inner integration steps alleviates the stability concern. Then a forward projection (extrapolation) is performed over a long step commensurate with the slow time constants from the results of the 'inner' integration without violating the stability constraint. The time step size of the outer projection step is chosen to track the slow components and set solely by (local) accuracy control. Because in the projection step, the solution at $t_{n+k+1+M}$ is extrapolated based on the solutions at t_{n+k} and t_{n+k+1} as

$$x_{n+k+1+M} = (M+1)x_{n+k+1} - Mx_{n+k}, \tag{5.1}$$

the accumulated error is only linearly amplified after it is exponentially damped in the preceding inner integration steps. As a result, the stability of projective integration method is maintained and the overall efficiency of projective integration is boosted by the outer projective step.

The projective integration idea is presented based on the simplified assumption that there exists only one gap between the time constants. For the more general conditions, the eigenvalues (or time constants) of the system may have more than one gap as shown in Fig.21 or be widely distributed without any obvious isolations between eigenvalue clusters, or even more the distribution is not known in advance. Under these cases, the step size of the outer projective step is significantly constrained ($M < 3k$) to ensure stability [73], which heavily deteriorates the efficiency of projective integration. To remedy this problem, a multi-level projective integration approach is suggested in [74]. The basic idea is that although at each level a limited speedup of $M + k + 1/k + 1$ is obtained when M is relatively small, a significant overall runtime speedup $(M + k + 1/k + 1)^q$ can be obtained in a q-level telescopic framework. Therefore, we can still maintain a good simulation efficiency without loss of stability. A telescopic projective integration framework is shown in Fig.22.

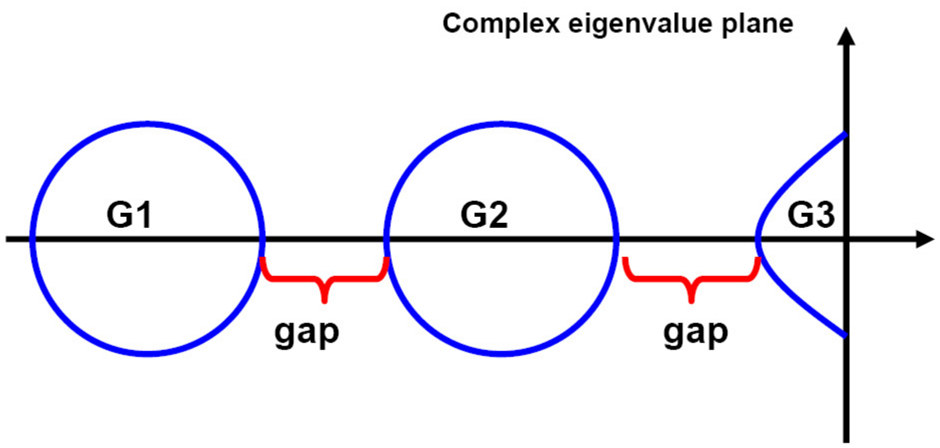

Fig. 21. Distribution of the eigenvalues with multiple gaps.

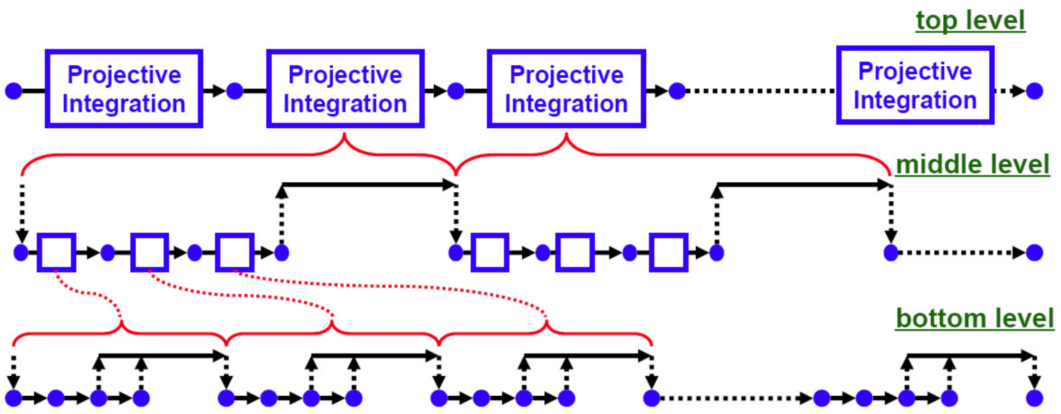

Fig. 22. Telescopic projective framework.

Correspondingly, the theoretical performance metrics for q level telescopic projective method is defined as :

$$\begin{aligned} \textbf{Efficiency Improvement} \ &\triangleq \ \left(\frac{M}{k+1}\right)^q \\ \textbf{Speedup} \ &\triangleq \ \left(\frac{M+k+1}{k+1}\right)^q \end{aligned} \quad . \tag{5.2}$$

As explained in [73, 74], the 'inner' integrator uses a small time step to damp the rapidly decaying components from a stability concern. Therefore, k should be properly selected for this purpose. For the selection of M, efficiency and accuracy should be balanced. It deserve mentioning that for large M, the single-level projective method is only stable for problems with a gap in their spectrum.

C. Stable explicit numerical integration for circuit simulation

Similar to (3.1), an electronic circuit can be described using differential equation in time domain as

$$F(X(t)) + \frac{d}{dt}Q(X(t)) + U(t) = 0, \tag{5.3}$$

where $X(t)$ is the vector of nodal voltages and branch currents, $U(t)$ is the input, $F(\cdot)$ and $Q(\cdot)$ are the functions describing static and dynamic nonlinearities. In parallel to the telescopic projective framework described in the previous section, a stable explicit numerical integration method is proposed for circuit simulation as shown in Fig. 23.

In the proposed explicit telescopic projective framework, Forward Euler is adopted as the 'inner' integrator at the bottom level of the hierarchical projective framework, as shown in Fig. 23. At each level of the telescopic projective loop, a combination of the inner explicit integration and the outer projection is employed. Because the projection step is limited without the knowledge of the exact eigenvalue distribution of the system in advance, a multi-level combination scheme is required to have an overall good runtime speedup.

Since the inner Forward Euler integrator and the outer projective integrator are both explicit, the entire integration scheme is explicit in nature. As will be seen in the following stability analysis, the presented integration scheme has good stability properties, which is crucial for practical circuit simulation applications.

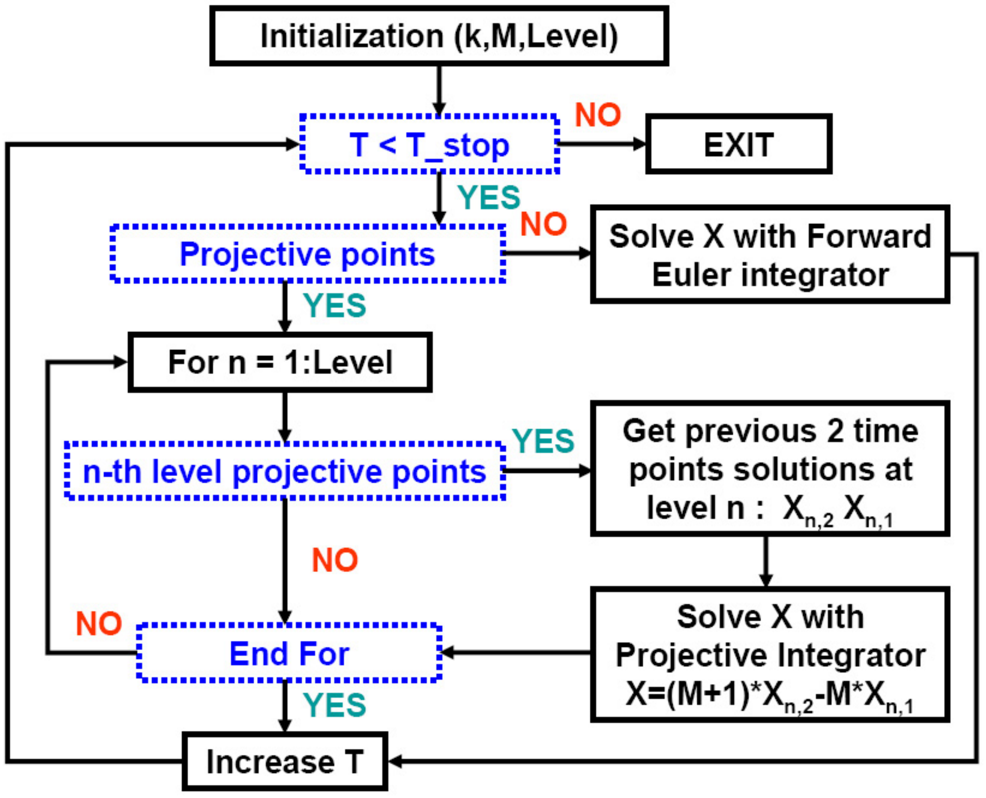
Initialization (k,M,Level)
T < T_stop
NO
EXIT
YES
Projective points
NO
Solve X with Forward Euler integrator
YES
For n = 1:Level
n-th level projective points
YES
Get previous 2 time points solutions at level n : $X_{n,2}$ $X_{n,1}$
NO
NO
End For
Solve X with Projective Integrator $X=(M+1)*X_{n,2}-M*X_{n,1}$
YES
Increase T

Fig. 23. Proposed stable explicit numerical integration for circuit simulation.

1. Stability of the standard Forward Euler

Without loss of generality, the following simpler linear system is used to analyze stability:

$$G \cdot X(t) + C \cdot \frac{dX(t)}{dt} + U(t) = 0, \tag{5.4}$$

where G and C are the linearized conductance and capacitance matrices. When Forward Euler integration is used, the equation (5.4) is converted into a set of discretized equations:

$$\begin{array}{l} G \cdot X(t_n) + C \cdot \frac{X(t_{n+1}) - X(t_n)}{h} + U(t_n) = 0 \\ \Rightarrow X(t_{n+1}) = C^{-1} \cdot [-hGX(t_n) + CX(t_n) - hU(t_n)] \end{array}. \tag{5.5}$$

Note that when Forward Euler is used for numerical integration, each circuit node is assumed to have a grounded capacitance. Otherwise, a small dummy capacitance will be inserted. With this, matrix C in the above is nonsingular for typical cases. According to the linear stability theory for an initial value ODE problem $dX/dt = -AX$ ($A = C^{-1}G$ for the linear circuit problem), the absolute stability region is $|1 - \lambda h| \leq 1$, where λ is an eigenvalue of A. Therefore, the time step should be $h < 2/\lambda_{max}$.

2. Stability of projective integration

Different from the analysis in [73, 74], the stability property is analyzed based on the modified nodal formulation and practical issues in circuit simulation is addressed. Since the numerical stability for linear circuit system equation (5.4) is not affected by $U(t)$, the following numerical stability analysis is based on its homogenous system

$$G \cdot X(t) + C \cdot \frac{dX(t)}{dt} = 0. \tag{5.6}$$

To solve the differential equation (5.6) numerically using the the projective integration method, several 'inner' steps using Forward Euler are combined with one 'outer' projective step. (5.6) is discretized to a sequence of algebraic equations at different time points, which would be solved by the 'inner' steps or 'out' steps. For the first 'inner' steps, based on Forward Euler integration, the algebraic equation at time point t_{n+i+1} can be represented as

$$G \cdot X_{n+i+1} + C \cdot \frac{X_{n+i+1} - X_{n+i}}{h} = 0, \tag{5.7}$$

where X_{n+i+1} and X_{n+i} are the solutions at time points t_{n+i+1} and t_{n+i}; h is the time step between time points t_{n+i+1} and t_{n+i}. Then the explicit relation between the solutions X_{n+i+1} and X_{n+i} is as follows: (In circuit simulation, the existence of the matrix inversion in the following derivation can be guaranteed.)

$$\begin{array}{l} G \cdot X_{n+i} + C \cdot \frac{X_{n+i+1} - X_{n+i}}{h} = 0 \\ \Rightarrow \quad \left(G - \frac{C}{h}\right) \cdot X_{n+i} + \frac{C}{h} \cdot X_{n+i+1} = 0 \\ \Rightarrow \quad X_{n+i+1} = (I - hC^{-1}G) \cdot X_{n+i} \end{array} . \tag{5.8}$$

Without loss of generality, for simplicity, assume that $C^{-1}G$ is diagonalizable and can be represented as $C^{-1}G = P\Lambda P^{-1}$. Therefore, equation (5.8) can be further written to:

$$\begin{array}{l} X_{n+i+1} = (I - hP\Lambda P^{-1}) \cdot X_{n+i} \\ \Rightarrow \quad X_{n+i+1} = P\left(I - h\Lambda\right) P^{-1} \cdot X_{n+i} \end{array} . \tag{5.9}$$

According to above equation, the solutions after k and $k+1$ 'inner' steps can be respectively represented as

$$\left\{ \begin{array}{c} X_{n+k} = P\left(I - h\Lambda\right)^{k} P^{-1} \cdot X_{n} \\ X_{n+k+1} = P\left(I - h\Lambda\right)^{k+1} P^{-1} \cdot X_{n} \end{array} \right. . \tag{5.10}$$

After $k+1$ 'inner' step, one 'outer' projective step would directly extrapolate the solution at time point $t_{n+k+1+M}$ based on the known solutions at time points t_{n+k} and t_{n+k+1}. (5.10) is substituted into (5.1) and reach the following relationship spanning one projective step:

$$X_{n+k+1+M} = P\left((M+1)(I-h\Lambda)^{k+1} - M(I-h\Lambda)^{k}\right)P^{-1}X_n. \tag{5.11}$$

Let $Y = P^{-1}X$, then

$$Y_{n+k+1+M} = \left((M+1)(I-h\Lambda)^{k+1} - M(I-h\Lambda)^{k}\right)Y_n. \tag{5.12}$$

Λ is a diagonal matrix and denote the i-th diagonal entry of Λ as λ_i. The corresponding components of $Y_{n+k+1+M}$ and Y_n are $y_{n+k+1+M,i}$ and $y_{n,i}$, respectively. Then

$$y_{n+k+1+M,i} = \left((M+1)(1-h\lambda_i)^{k+1} - M(1-h\lambda_i)^{k}\right)y_{n,i}. \tag{5.13}$$

To guarantee the stability, $y_{n+k+1+M,i}$ and $y_{n,i}$, one must meet the following condition:

$$\begin{array}{l} |\phi| = \left|\frac{y_{n+k+1+M,i}}{y_{n,i}}\right| \leq 1 \\ \Rightarrow \left|(M+1)(1-h\lambda_i)^{k+1} - M(1-h\lambda_i)^{k}\right| \leq 1 \\ \overset{\gamma_i=1-h\lambda_i}{\Rightarrow} \left|(M+1)\gamma_i^{k+1} - M\gamma_i^{k}\right| \leq 1 \end{array}. \tag{5.14}$$

Note that γ_i is complex number. Express $\gamma_i = 1 - h\lambda_i = x + jy = re^{j\varphi}$. Then

$$\begin{array}{l} |\phi|^2 = \left|\frac{y_{n+k+1+M,i}}{y_{n,i}}\right|^2 \\ = (M+1)^2 r^{2(k+1)} + M^2 r^{2k} - 2(M+1)Mr^{2k+1}\cos\varphi \\ = \left((M+1)^2(x^2+y^2) + M^2 - 2(M+1)Mx\right)(x^2+y^2)^k. \end{array} \tag{5.15}$$

The stability region can be found by plotting the locus of all $h\lambda$ for which $|\phi| = 1$. Therefore the coordinates $(\hat{x}, \hat{y})$ of $h\lambda$ in complex $h\lambda$ plane have the following relation.

$$\begin{array}{l} h(\hat{x},\hat{y}) = (M^2 - 2(M+1)M(1-\hat{x}))\left((1-\hat{x})^2 + \hat{y}^2\right)^k \\ + (M+1)^2\left((1-\hat{x})^2 + \hat{y}^2\right)^{k+1} - 1 = 0 \end{array}. \tag{5.16}$$

In the projective integration, a projection (extrapolation) step based on equation (5.1) is executed after several inner integrations. As an alternative of the projection step, a Forward Euler with a larger time step compared with the inner step can be adopted. The the stability can be analyzed as follows:

$$\begin{aligned}
&G \cdot X_{n+k+1} + C \cdot \tfrac{X_{n+k+1+M} - X_{n+k+1}}{Mh} = 0 \\
&\Rightarrow X_{n+k+1+M} = -\left(\tfrac{C}{Mh}\right)^{-1}\left(G - \tfrac{C}{Mh}\right) X_{n+k+1} \\
&\Rightarrow X_{n+k+1+M} = \left(I - MhC^{-1}G\right) X_{n+k+1} \\
&\Rightarrow X_{n+k+1+M} = P\left(I - Mh\Lambda\right) P^{-1} X_{n+k+1} \\
&\Rightarrow X_{n+k+1+M} = P\left(I - Mh\Lambda\right) P^{-1} P\left(I - h\Lambda\right)^{k+1} P^{-1} X_n \\
&\Rightarrow P^{-1} X_{n+k+1+M} = \left(I - Mh\Lambda\right)\left(I - h\Lambda\right)^{k+1} P^{-1} X_n \\
&\Rightarrow Y_{n+k+1+M} = \left(I - Mh\Lambda\right)\left(I - h\Lambda\right)^{k+1} Y_n
\end{aligned} \quad . \tag{5.17}$$

Then the i^{th} component of vector $Y_{n+k+1+M}$ can be represented as

$$\begin{aligned}
&y_{n+k+1+M,i} = \left(1 - Mh\lambda_i\right)\left(1 - h\lambda_i\right)^{k+1} y_{n,i} \\
&\Rightarrow y_{n+k+1+M,i} = \left(1 - (M+1)h\lambda_i + Mh^2\lambda_i^2\right)\left(1 - h\lambda_i\right)^{k} y_{n,i}(*)
\end{aligned} \quad . \tag{5.18}$$

If the high order term $Mh^2\lambda_i^2$ in equation $(*)$ is omitted, then it is the same as equation (5.13) as shown in the following equation

$$\begin{aligned}
&y_{n+k+1+M,i} = \left(1 - (M+1)h\lambda_i + Mh^2\lambda_i^2\right)\left(1 - h\lambda_i\right)^{k} y_{n,i} \\
&\Rightarrow y_{n+k+1+M,i} \approx \left((M+1)\left(1 - h\lambda_i\right) - M\right)\left(1 - h\lambda_i\right)^{k} y_{n,i}
\end{aligned} \quad , \tag{5.19}$$

which means that their stability regions are the same as well.

3. Stability of telescopic projective integration

The telescopic projective method can be understood as a multi-level projective method in essence, therefore its stability can be analyzed based on the stability of the one-level projective method. Without loss of generality, consider one step of second-level 'outer' integrator. Equation (5.11) can be simply written as

$$X_{n+(k+1+M)} = P\Phi P^{-1}X_n, \tag{5.20}$$

where $\Phi = (M+1)\left(I - h\Lambda\right)^{k+1} - M\left(I - h\Lambda\right)^{k}$. Then

$$\begin{aligned} & X_{n+(k+1+M)^2} = (M+1)\,X_{n+(k+1)(k+1+M)} - MX_{n+k(k+1+M)} \\ & \Rightarrow X_{n+(k+1+M)^2} = P\left((M+1)\,\Phi^{k+1} - M\Phi^{k}\right)P^{-1}X_n \\ & \overset{Y=P^{-1}X}{\Rightarrow} Y_{n+(k+1+M)^2} = \left((M+1)\,\Phi^{k+1} - M\Phi^{k}\right)Y_n \end{aligned} . \tag{5.21}$$

$$\phi_{telescopic} = \frac{y_{n+(k+1+M)^2,i}}{y_{n,i}} \overset{\gamma_i=1-h\lambda_i}{=} (M+1)\,\phi^{k+1} - M\phi^{k}. \tag{5.22}$$

Similar to projective method, the stability region is defined as $|\phi_{telescopic}| \leq 1$. It can be proven that there exists a [0, 1] stability region for telescopic projective method [74], which implies that the stability region includes all of the real axis in complex γ_i plane. Since the 'inner' integrator is Forward Euler method $\gamma_i = 1 - h\lambda_i$ in the proposed method, it means that for any real $\lambda_i \in \left[0, \frac{1}{h}\right]$, the telescopic projective method can guarantee the stability with the certain parameters k and M.

4. Parallel implementation

The principle and the stability issues of the proposed explicit telescopic projective method has been demonstrated in detail. In this section, some important issues for the implementation of the proposed method are explained.

With the use of explicit numerical integration, it is desirable to integrate the transient circuit response on a per-node or per-device basis without solving any coupled large systems of equations. This leads to natural parallelization. As shown in Fig. 24 (a), while this goal is straightforward to achieve when there exists no coupling between different circuit nodes, complication arises if coupling does exists, as illustrated in Fig. 24 (b). In the latter case, capacitance currents i_{c1} and i_{c2}, which are needed in explicit numerical integration, can no longer be determined individually at each circuit node. Instead, a coupled system involving both nodes needs to be solved.

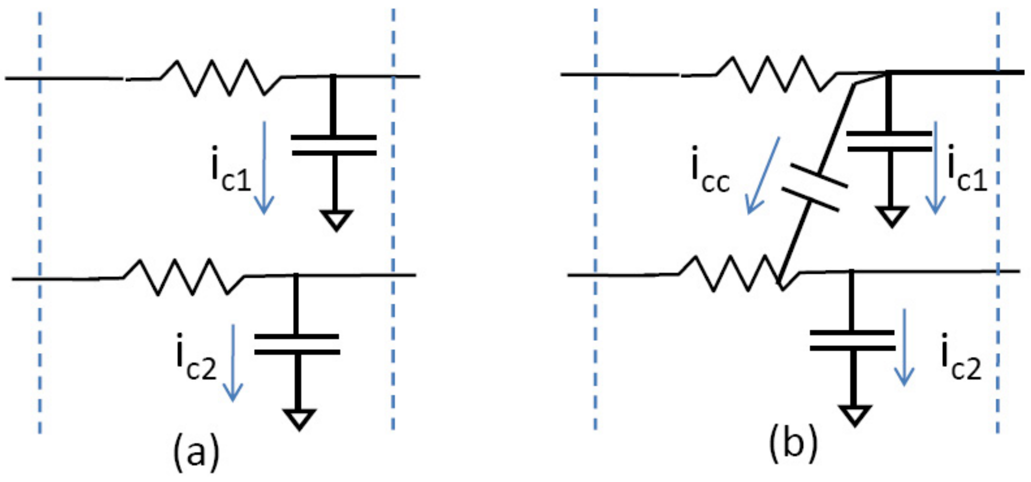

Fig. 24. Two circuit nodes: (a) without coupling, and (b) with coupling.

Since Forward Euler is used as the 'inner' integrator, solving a linear matrix problem at each time step can be avoided compared with those using implicit integration methods and the transient circuit response can be obtained on a per-node or per-device basis as shown in Fig. 24 (a), which leads to natural parallelization. But it is not always true when Forward Euler analysis in equation (5.5) is considered. Actually, only if the capacitance matrix C is diagonal, solving the linear matrix equation is avoided. Each component in solution vector $X(t_{n+1})$ can be solved in a row based way. Physically it means that there only exist the grounded capacitors in the circuit. The slope of each voltage waveform is determined by computing the branch current of each grounded capacitor. And accordingly, the node voltage is projected to be:

$$V(t_{n+1}) = \frac{i_{branch} \cdot (t_{n+1} - t_n)}{C} + V(t_n). \tag{5.23}$$

But when C is a non-diagonal matrix, the equation (5.5) cannot be solved in a row-based way. Physically, the case corresponds to the situation where the circuit has coupling capacitances as illustrated in Fig. 24 (b). In this case, capacitance currents i_{c1} and i_{c2}, which are needed in explicit numerical integration, can no longer be determined individually at each circuit node. Instead, a coupled system involving both nodes needs to be solved. To avoid solving any matrix problem, C_n is split into a diagonal matrix part Λ_n and a off-diagonal matrix part N_n similar to the approach in [71]:

$$\begin{array}{l} G_n V(t_n) + C_n \frac{V(t_{n+1}) - V(t_n)}{h} + U(t_n) = 0 \\ \Rightarrow \;\; G_n V(t_n) + (\Lambda_n + N_n) \frac{V(t_{n+1}) - V(t_n)}{h} + U(t_n) = 0 \\ \Rightarrow \;\; N_n \frac{V(t_n) - V(t_{n-1})}{h} \approx N_n \frac{V(t_{n+1}) - V(t_n)}{h} \\ \quad = -G_n V(t_n) - \Lambda_n \frac{V(t_{n+1}) - V(t_n)}{h} - U(t_n) \\ \Rightarrow \;\; V(t_{n+1}) \approx \Lambda_n^{-1} [-h G_n V(t_n) + \Lambda_n V(t_n) - N_n V(t_n) \\ \quad + N_n V(t_{n-1}) - h U(t_n \;\;)] \end{array} . \qquad (5.24)$$

Essentially, the branch currents at the preceding time point are employed to solve the branch currents that go into each grounded capacitance at the present time point.

Another important issue is the handling of the parasitics of nonlinear devices such as MOSFET. For example, a MOSFET is considered as a four-terminal device and the nonlinear gate capacitance is modeled by specifying the coupled charge equations at the four terminals. The branch currents must be decided by solving the four coupled equations together. This implies that multiple small nonlinear systems of equations need to be solved, which doesn't present any computational challenge.

By properly handling the issues above, the relative independence for updating the voltage waveform at each circuit node is maintained. Therefore, the proposed FE based telescopic projection can be parallelized very straightforwardly. The flow

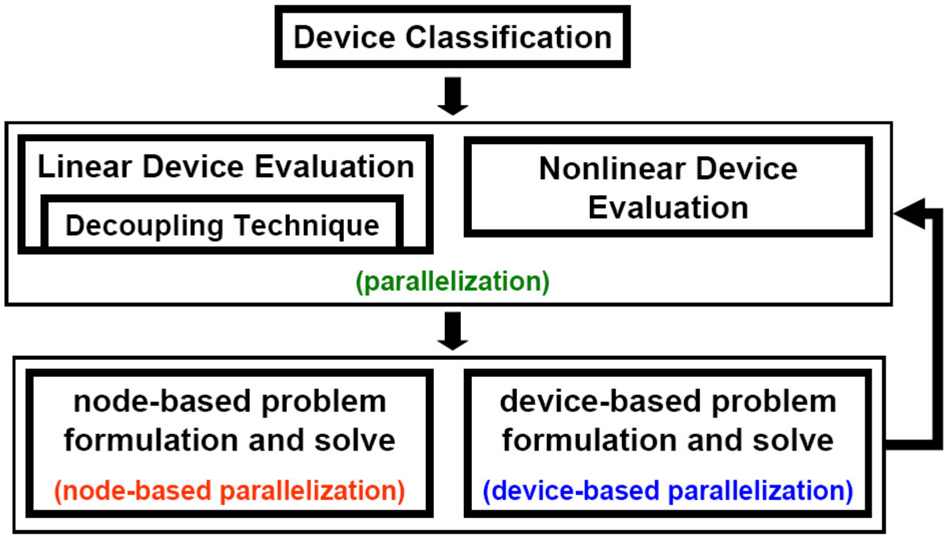
Device Classification
Linear Device Evaluation
Decoupling Technique
Nonlinear Device Evaluation
(parallelization)
node-based problem formulation and solve
(node-based parallelization)
device-based problem formulation and solve
(device-based parallelization)

Fig. 25. Parallel simulation framework.

of such parallel simulation is shown in Fig.25.

D. Experimental results

In order to validate the proposed idea, the explicit telescopic projective integration method in a SPICE-like simulator is implemented using C/C++. And Pthreads is also used to support multithread programming in the multicore platform.

1. Accuracy and efficiency

In this section, the accuracy and the efficiency for the proposed numerical integration method is firstly verified. Without loss of generality, some simple RC circuits are selected to demonstrate the benefits of the new method.

It is known that the disadvantage of Forward Euler method is that the simulation time step is limited by the minimum eigenvalue due to the stability issue. Considering the test circuit in Fig.26, the time step is restricted by the capacitor with the capacitance value $1f$F. Therefore, a proper time step is on the order of 10^{-15} second for Forward Euler integration. For the input shown in Fig.27, 10^5 time points need to run in total.

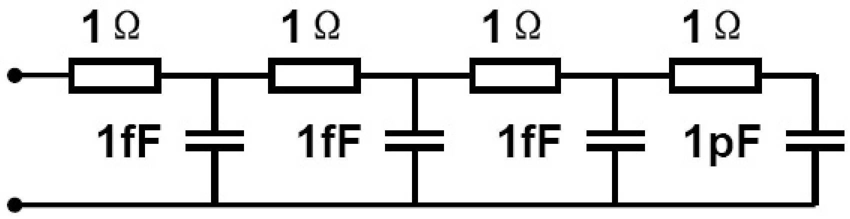

Fig. 26. Stiff RC circuit 1.

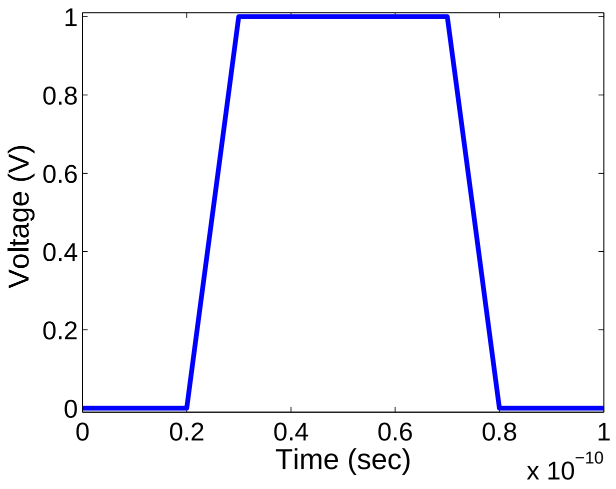

Fig. 27. Input waveform.

The circuit is stiff due to the different scales of the capacitance values between $1f$F and $1p$F and the eigenvalues of the ordinary differential equation defined by the circuit system are clustered into two groups. If stability is the main concern, by using the proposed method, a significant time-step amplification is achieved. Assume that one-level telescopic projective method is used and Forward Euler is adopted as the 'inner' integrator. Compared with Forward Euler method, the theoretical time-step amplification is $(M + k + 1)/(k + 1) = 51$ if $k = 9$ and $M = 500$; Correspondingly if $k = 4$ and $M = 500$, the time-step amplification is 101. The total number of time points are 2167 and 1194 respectively. In Fig.28, the output waveforms for the node connected to $1p$F capacitor are shown. In the figure, 'BE' means the waveform simulated by using Backward Euler method; 'FE' corresponds to Forward Euler method; 'TP1' and 'TP2' represent the results using the projective methods. From the results, it can be seen that the waveforms for the projective methods are well-matched to that for the standard Backward Euler method. It can also be observed that the accuracy is directly relevant to the values of k and M, which can be controlled using LTE (Local Truncation Error).

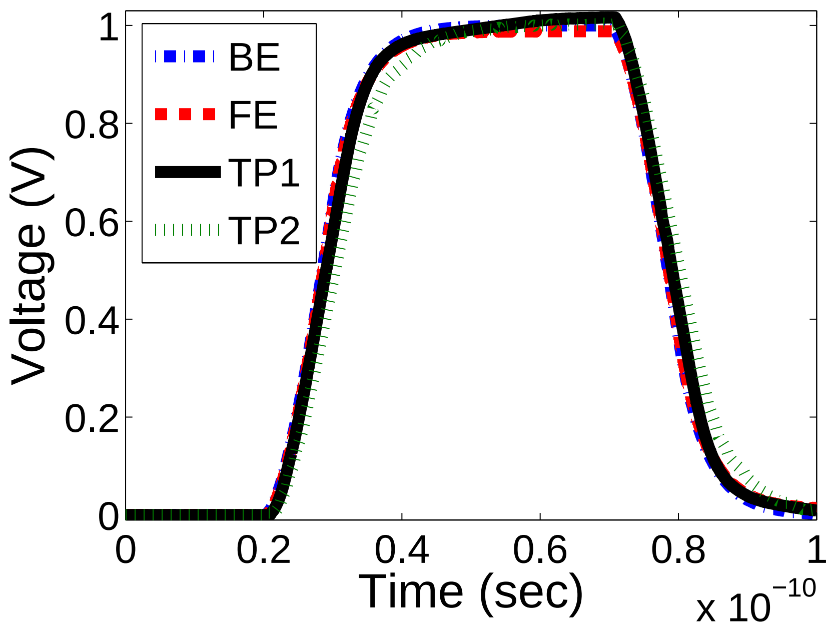

Fig. 28. Transient simulation for circuit 1.

In the second test circuit as shown in Fig. 29, there exist three far-different scaled capacitors. Since the minimum capacitance is still $1f$F, the time step for Forward Euler simulation to ensure the stability is similar to the first test case. The two-level telescopic projective method is adopted in this experiment. If $k = 9$ and $M = 20$, the total theoretical time-step amplification is $(M + k + 1/k + 1)^2 = 9$; and the theoretical time-step amplification is 16 if $k = 9$ and $M = 30$. In Fig. 30, the output waveforms for the node connected to 10pF capacitor are shown. In the figures, 'TP1' and 'TP2' represent the results using the two-level telescopic projective methods. The time steps of BE, FE and that of the inner integrator in telescopic projective integration are set to be the same. It can be seen that the waveforms for the telescopic projective methods are also well-matched to that for the standard Backward Euler method.

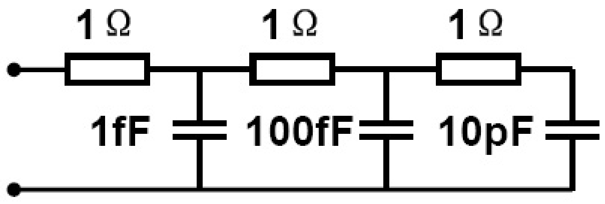

Fig. 29. Stiff RC circuit 2.

2. Serial and parallel simulation

Next, we apply backward Euler, two-level serial explicit telescopic projective integration, and its two-thread and four-thread parallel versions to a number of test circuits. For the two-level telescopic method, we set $k = 3$ and $M = 1$. The runtime statistics are collected in Table VIII. In the table, *Time* is the total runtime and *Speedup* represents the speedup over backward Euler method. When the number of threads is low (one or two), telescopic integration can be actually slower than BE. This is not very surprising since in telescopic integration multiple inner forward Euler integration steps are needed to ensure stability. However, because of the explicit nature of the

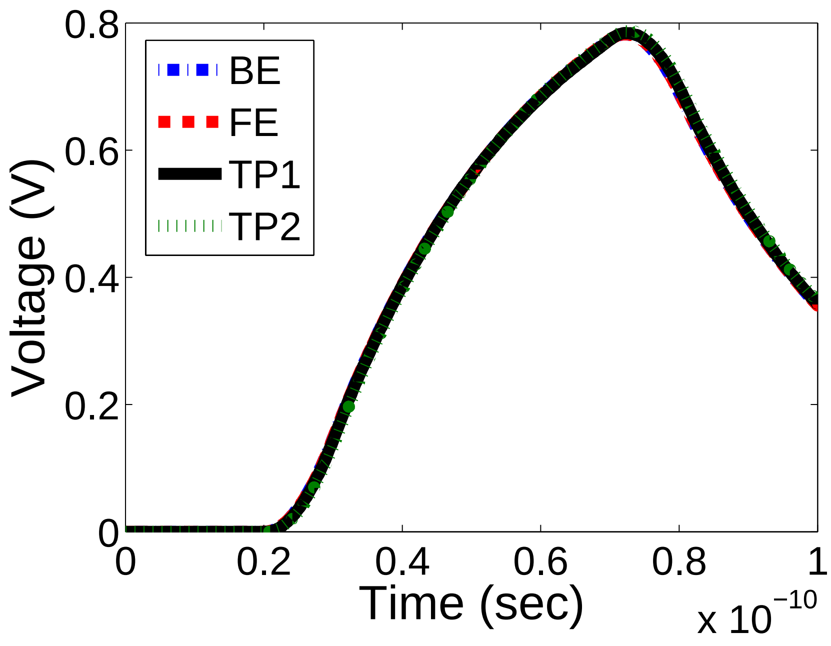

Fig. 30. Transient simulation for circuit 2.

method, parallelisms can be easily exploited by adding more threads to gain runtime benefits. This would be particularly meaningful for large circuits, where potentially a large number of threads can be executed currently to process the large workload.

Table VIII. Statistics of the transient simulations on serial and parallel platforms.

Circuit	Serial		2-thread	4-thread
	BE	Proposed	Proposed	Proposed
	Time(s)	Speedup	Speedup	Speedup
Buffer chain	36	0.54	0.72	1.17
DB mixer	33	0.61	0.87	1.52
4-bit adder	207	0.73	1.18	1.79
RC mesh 1 w/drivers	582	1.06	1.79	2.84
RC mesh 2 w/drivers	2,611	2.15	3.60	5.67

E. Summary

In this chapter, an explicit telescopic integration method is proposed for transient simulation, especially for some stiff circuit problems, since Forward Euler integration is adopted as the 'inner' integrator in the telescopic projective framework, the proposed method guarantees the stability of the overall integration scheme. At the same time, the explicit nature of the proposed method can be exploited to speed up transient simulation via efficient parallelization.

CHAPTER VI

PARALLEL HARMONIC BALANCE SIMULATION

A. Introduction

As is demonstrated in Chapter III, HB analysis is a steady-state simulation technique in frequency domain for periodic and quasi-periodic responses [60]. Due to large and densely-coupled systems of nonlinear equations in HB problem formulation, speeding up HB analysis via parallel computing is meaningful, especially for the design of wide range of analog and RF ICs. Although, some efforts on parallel techniques have been proposed in the past to facilitate HB analysis (e.g. [44–47, 75–77]), more efficient and robust parallel HB simulation techniques are in demand by addressing the limitations of existing methods.

In this chapter, a parallel HB analysis approach is proposed. This approach is centered on parallelizing one of the key computational steps of HB : preconditioning, which not only determines the efficiency and robustness of the simulation, but also corresponds to a fairly significant portion of the overall computing work. For these reasons, a parallel HB approach is developed based on the hierarchical preconditioning technique in [64, 65]. Under the context of preconditioning, by recursively partitioning the linearized HB problem into a series of smaller independent matrix problems across multiple levels, a tree-like data dependency structure is resulted. This naturally provides a coarse-grained parallelization opportunity that is being investigated in this research.

Compared with the parallelization of the standard BD preconditioning in [47], the proposed approach has several advantages. Firstly, the improved efficiency and robustness of the hierarchal preconditioner [64, 65] over the BD preconditioner is naturally carried over, better contributing to the performance of parallel HB simulation. Secondly, since the use of the hierarchical preconditioner pushes more computational work towards the preconditioning, making an efficient parallel preconditioner more appealing. Lastly, the parallelization of the standard BD preconditioner is on a pre-determined per-frequency basis where the parallelization granularity, which is identical to the size of a diagonal block, is fixed. The tree-like structure of the hierarchical preconditioner, on the other hand, provides more freedom in choosing suitable parallelization granularity to fit a given parallel hardware system.

Furthermore, a unified parallel simulation framework is developed based on the same parallel preconditioning principle, which is applicable not only to the steady-state analysis of driven circuits, but also to that of autonomous circuits and to the envelope-following analysis. The proposed simulation framework admits straightforward integration of traditional parallelizing ideas such as parallel device model evaluations, parallel FFT/IFFTs and parallel matrix-vector products. For the three types of the analyses above, all favorable runtime speedups are achieved in the message-passing-interface (MPI) based implementations over a cluster of workstations and multi-threading based implementations on a shared-memory machine with respect to not only the traditional serial simulation algorithms but also the serial implementation of the same proposed algorithms.

In the rest of this chapter, the principle of the parallel preconditioning based HB method is illustrated in details first. Then, the basic parallel HB idea is extended to accommodate the steady-state analysis of autonomous circuits and the envelope-following analysis. In the next section, the important parallel programming implementation issues are discussed. The numerical experimental results are presented in Section E. Finally, a summary of this chapter is given.

B. Proposed parallel HB analysis

To identify possible ways to parallelize HB, the standard flow of HB simulation is reviewed first as shown in Fig.31. It can be seen that the device model evaluation

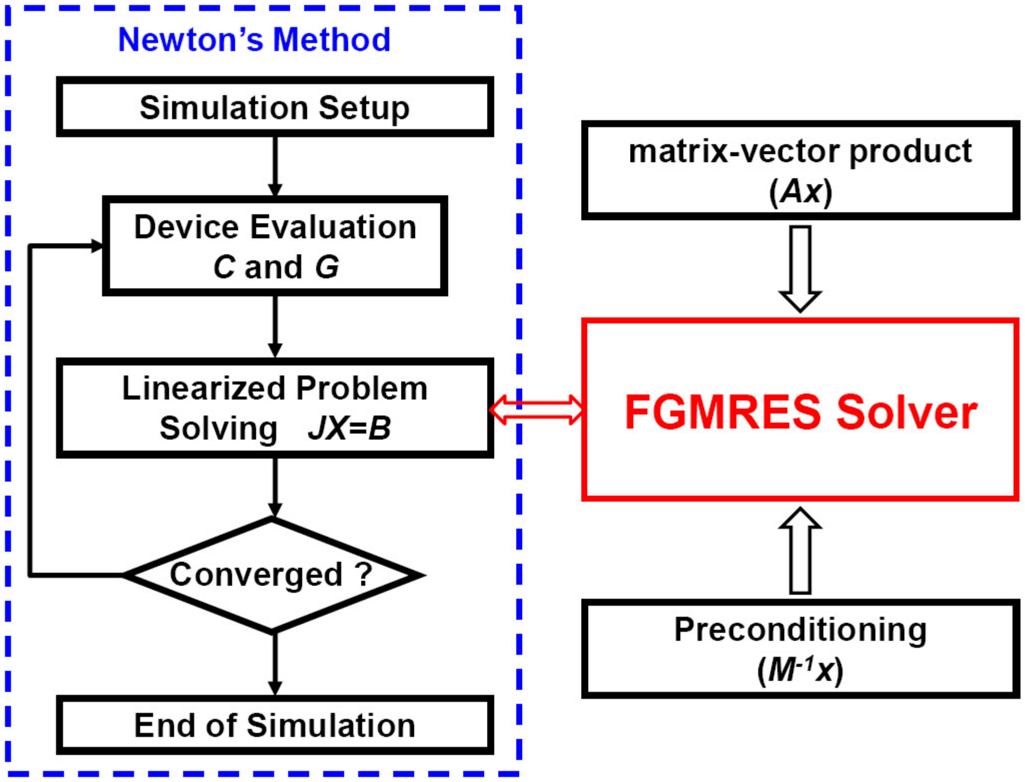
Newton's Method
Simulation Setup
Device Evaluation C and G
Linearized Problem Solving JX=B
Converged ?
End of Simulation
matrix-vector product (Ax)
FGMRES Solver
Preconditioning ($M^{-1}x$)

Fig. 31. A basic flow for HB analysis.

and the linearized problem-solving are the two basic steps at each Newton iteration. In Fig. 32, the detailed task dependency is shown.

Parallelizing the device model evaluation is fairly straightforward, which can be done by running multiple device model evaluations across several processing elements (PEs). Usually, a near-linear runtime scaling can be achieved. Matrix-vector product and preconditioning are the two key operations in solving the linearized HB problem during each Newton iteration. In HB, matrix-vector products associated with

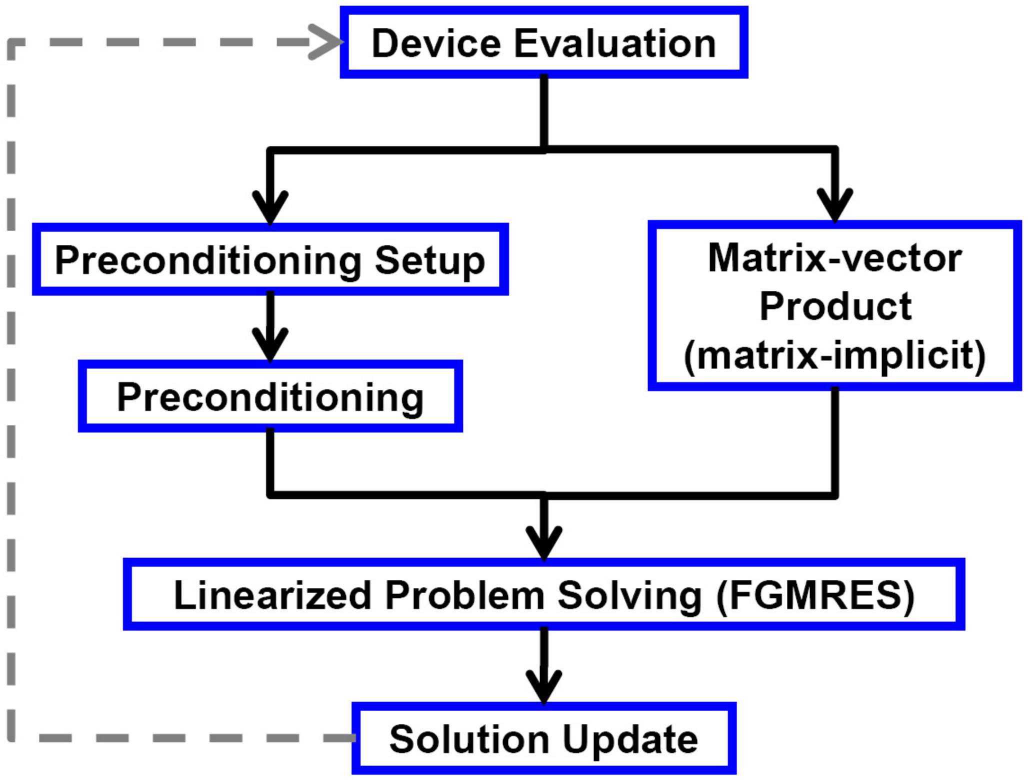
Device Evaluation
Preconditioning Setup
Preconditioning
Matrix-vector Product (matrix-implicit)
Linearized Problem Solving (FGMRES)
Solution Update

Fig. 32. Task dependency of the operations in each Newton iteration for HB analysis.

Jacobian matrix J in (3.6) are needed

$$JX = \Omega(\Gamma(C(\Gamma^{-1}X))) + \Gamma(G(\Gamma^{-1}X)), \tag{6.1}$$

where Ω is a diagonal matrix representing the frequency domain differentiation operator; Γ and Γ^{-1} are the N-point FFT and IFFT matrices;$C = diag\{c_k = \frac{\partial q}{\partial x}|_{x=x(t_k)}\}$ and $G = diag\{g_k = \frac{\partial f}{\partial x}|_{x=x(t_k)}\}$ are block-diagonal matrices with the diagonal blocks representing the linearizations of $q(\cdot)$ and $f(\cdot)$ at N sampled time points $t_1, t_2, \cdots, t_N$.

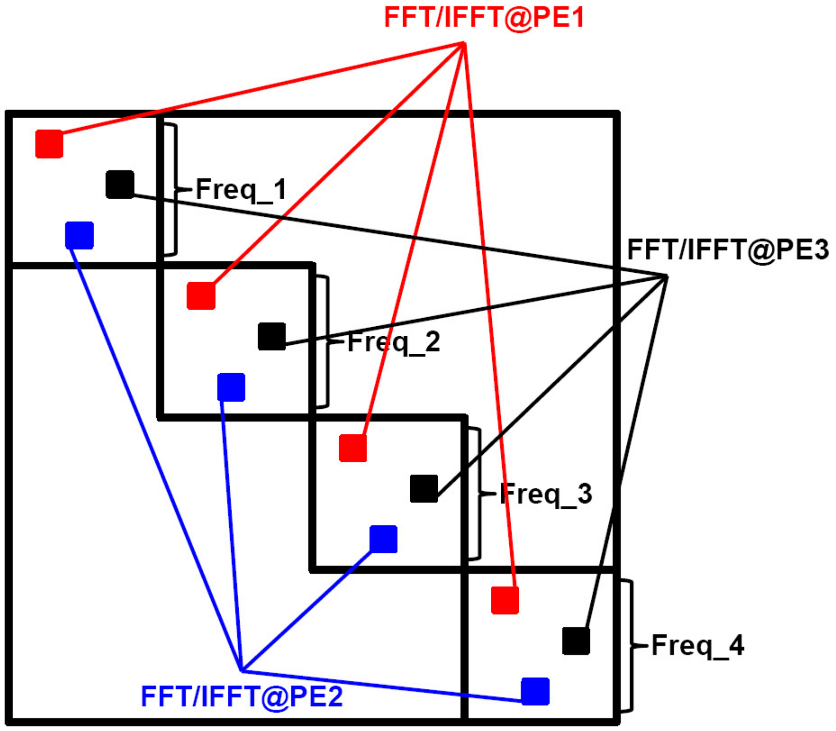

Fig. 33. Parallelization of FFT/IFFT operations.

Since the matrix vector product can be efficiently achieved by FFT/IFFT operations, it can be accelerated by parallelizing the basic FFT/IFFT operations. Considering that the same FFT/IFFT operations should be independently applied to every signal entries, a straightforward data parallelism approach can be used to simultaneously executed the multiple FFT/IFFT operations on the different input data as shown in Fig. 33. Similarly, because the low-level matrix computations are organized in a 'for-loop' structure, it is not difficult to be parallelized as well.

In comparison, adopting and parallelizing an effective preconditioner, which is not only efficient and robust but also flexible in parallel processing, is more involved. this issue is focused in the remainder of this section.

1. Basic ideas of parallel hierarchical preconditioning

To construct a parallel preconditioner to solve the linearized problem $JX = B$ defined by (6.1), the parallelizable operations that are involved should be identified. Assuming that there are totally m PEs available, (6.1) is rewritten as

$$\begin{bmatrix} J_{11} & J_{12} & \cdots & J_{1m} \\ J_{21} & J_{22} & \cdots & J_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ J_{m1} & J_{m2} & \cdots & J_{mm} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix}, \tag{6.2}$$

where Jacobian J is composed of $m \times m$ block entries; X and B are correspondingly partitioned into m segments along the frequency boundaries. Further, J can be expressed in a form

$$[J]_{m\times m} = \left(\begin{bmatrix} \Omega_1 & & & \\ & \Omega_2 & & \\ & & \ddots & \\ & & & \Omega_m \end{bmatrix} C_c + G_c \right), \tag{6.3}$$

where circulants C_c, G_c are correspondingly partitioned as

$$\begin{aligned} C_c = \quad & \Gamma C \Gamma^{-1} = \begin{bmatrix} C_{c11} & \cdots & C_{c1m} \\ \vdots & \ddots & \vdots \\ C_{cm1} & \cdots & C_{cmm} \end{bmatrix} \\ G_c = \quad & \Gamma G \Gamma^{-1} = \begin{bmatrix} G_{c11} & \cdots & G_{c1m} \\ \vdots & \ddots & \vdots \\ G_{cm1} & \cdots & G_{cmm} \end{bmatrix} \end{aligned} . \qquad (6.4)$$

Because designing a parallel preconditioning for linearized problem $JX = B$ is essentially equivalent to find a parallel routine to approximately calculate JX, it can be started from find an approximation P to J. Assuming that the preconditioner is going to be parallelized using m PEs, the off-diagonal blocks of (6.4) are discarded, an approximation P to J can be obtained as shown in (6.5)

$$\begin{aligned} J = & \begin{bmatrix} \Omega_1 & & \\ & \ddots & \\ & & \Omega_m \end{bmatrix} \begin{bmatrix} C_{c11} & \cdots & C_{c1m} \\ \vdots & \ddots & \vdots \\ C_{cm1} & \cdots & C_{cmm} \end{bmatrix} + \begin{bmatrix} G_{c11} & \cdots & G_{c1m} \\ \vdots & \ddots & \vdots \\ G_{cm1} & \cdots & G_{cmm} \end{bmatrix} \\ \approx & \begin{bmatrix} \Omega_1 & & \\ & \ddots & \\ & & \Omega_m \end{bmatrix} \begin{bmatrix} C_{c11} & & \\ & \ddots & \\ & & C_{cmm} \end{bmatrix} + \begin{bmatrix} G_{c11} & & \\ & \ddots & \\ & & G_{cmm} \end{bmatrix} \\ = & P \end{aligned} , \qquad (6.5)$$

which leads to m decoupled linearized problems of smaller dimensions in (6.6)

$$\left\{ \begin{aligned} J_{11}X_1 &= [\Omega_1 C_{c11} + G_{c11}]X_1 = B_1 \\ J_{22}X_2 &= [\Omega_2 C_{c22} + G_{c22}]X_2 = B_2 \\ &\vdots \\ J_{mm}X_m &= [\Omega_m C_{cmm} + G_{cmm}]X_m = B_m \end{aligned} \right. . \qquad (6.6)$$

By solving these decoupled linearized problems in a parallel way, a parallel preconditioner is efficiently provided.

This basic idea of divide-and-conquer can be extended in a hierarchical fashion as shown in Fig. 34. At the topmost level, to solve the top-level linearized HB problem, a preconditioner is created by approximating the full Jacobian using a number (in this case two) of *super* diagonal blocks, which are shown in black. The partitioning of the full Jacobian is along the frequency boundary. These super blocks can be large in size so that it is difficult to solve them directly. Therefore, an iterative method such as FGMRES is again applied to each block problem, for which a preconditioner is further required. These preconditioners are created in the same fashion as that of the top-level problem by recursively decomposing a large block into smaller ones until the block size is small enough for a direct solve. This entire process leads to a multi-level hierarchical preconditioner for the original linearized HB problem. To avoid creating explicit representations for all the subproblems across the hierarchy, a matrix-implicit formation is adopted to save the memory usage, where low-pass filtered time-domain device equation linearizations are used to implicitly form the subproblems [64, 65]. The subproblems at the same tree depth are completely independent, hence they can be solved simultaneously. The hierarchical preconditioner is naturally parallelizable by algorithm construction, where the granularity of parallelization is controlled by either adjusting the sizes of the subproblems or the mapping from the tree structure to the actual parallel implementation on the hardware.

From an algorithmic point of view, the hierarchical preconditioner is more advantageous over the standard BD preconditioner. It provides a better approximation to the Jacobian, hence leading to improved efficiency and robustness, especially for strongly nonlinear circuits. On the other hand, from a parallel computing point of view, a parallel version of the hierarchical preconditioner provides a richer set of flex-

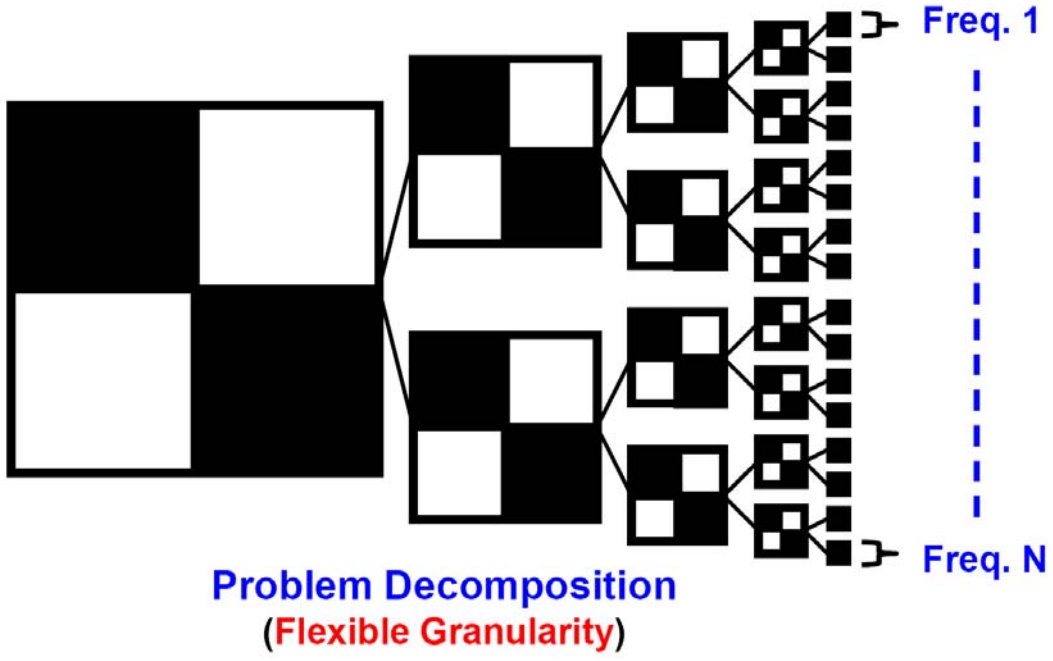

Fig. 34. Tree-like problem decomposition for the hierarchical preconditioner.

ibilities and tradeoffs than its BD preconditioner counterpart. While the granularity of the parallel BD preconditioner is pre-fixed, corresponding to the size of sub-matrix blocks for an individual frequency component, the tree-like structure of the hierarchical preconditioner can be altered to balance between the robustness and the efficiency by tuning the parallelization granularity. For instance, the number of levels and the number of subproblems at each level can be tuned for the best runtime performance; in addition, all the subproblems can be sized to create vertical computing task clusters with varying size and coupling intensity. In this way, a suitable hierarchical preconditioner may be constructed to fit a parallel hardware system with a specific number of PEs, where these PEs may differ in computing power and inter-PE communication overheads may vary within the system.

2. Analysis of runtime complexity and parallel efficiency

Denote M as the number of harmonics, N as the number of circuit nodes, K as the number of levels in the hierarchical preconditioner, P_i as the total number of sub-problems at level i ($P_1 = 1$ for the topmost level), and $I_{F,i}$ as the maximum number of FGMRES iterations required to reach the convergence for a sub-problem at level i. $S_{F,i} = \Pi_{k=1}^{i} I_{F,k}$, $i = 1, \cdots, K$ and $S_{F,0} = 1$ are defined.

The runtime cost in solving a sub-problem at the ith level can be broken into two parts: c1) the cost incurred by the FGMRES algorithm; and c2) the cost due to the preconditioning. In the serial implementation, the cost c1 at the topmost level is given by: $\alpha I_{F,1} MN + \beta I_{F,1} MN \log M$, where α, β are certain constants. The first term in c1 corresponds to the cost incurred within the FGMRES solver, which is linear assuming that the restart parameter is much smaller than $I_{F,i}$. The second term in c1 represents the cost of FFT/IFFT operations. At the topmost level, the cost c2 comes from solving P_2 sub-problems at the second level $I_{F,1}$ times, which is further equal to the cost of solving all the sub-problems starting from the second level in the hierarchial preconditioner. Adding everything together, the total runtime cost (also can be considered as computational complexity) of the serial hierarchically-preconditioned HB is

$$T_s = MN \sum_{i=1}^{K-1} P_i S_{F,i-1} \left(\alpha + \beta \log \frac{M}{P_i} \right) + \gamma S_{F,K} M N^{1.1}, \tag{6.7}$$

where the last term is due to the direct solve of the diagonal blocks of size N at the bottom of the hierarchy. It is assumed that directly solving a $N \times N$ sparse matrix problem has a cost of $O(N^{1.1})$.

For the parallel implementation, assume that the work load is evenly split among m PEs and the total inter-PE communication overhead is T_{comm}, which is proportional to the number of inter-PE communications. Correspondingly, the runtime cost for the parallel implementation is

$$T_p \; = \; \frac{MN\sum_{i=1}^{K-1} P_i S_{F,i-1}\left(\alpha + \beta \log \frac{M}{P_i}\right) + \gamma S_{F,K} MN^{1.1}}{m} + T_{comm}.$$

It can be seen that minimizing the inter-PE communication overhead (T_{comm}) is important in order to achieve a good parallel processing efficiency factor. The proposed hierarchical preconditioner is parallelized by simultaneously computing large chunks of independent computing tasks on multiple processing elements. The coarse-grain nature of the propsed parallel preconditioner reduces the inter-PE communication overhead and contributes to good parallel processing efficiency.

3. Processing element allocation

As discussed above, the tree-like task dependency of the hierarchical preconditioner in Fig. 35, makes it naturally parallelizable. In this subsection, it is discussed how to map a tree-like hierarchical preconditioner onto a parallel hardware, i.e., PE allocation.

First, consider a simple case, where the PEs have the identical computing powers and each problem is split into N equally-sized sub-problems at the next level in the hierarchical preconditioning. The PE allocation problem is defined to be the one that assigns a set of P PEs to n computing tasks so that the workload is balanced and there is no deadlock. The breadth-first traversal of the task dependency tree is used to allocate PEs, as shown in Algorithm 1.

The complete PE assignment can be determined by calling $Allocate(root, P_{all})$, where the *root* is the node representing the top-level linearized HB problem, and P_{all} is the full set of PEs. Two examples of PE allocations are shown in Fig. 36 for the cases of three and nine PEs available, respectively. In the three-PE case, the

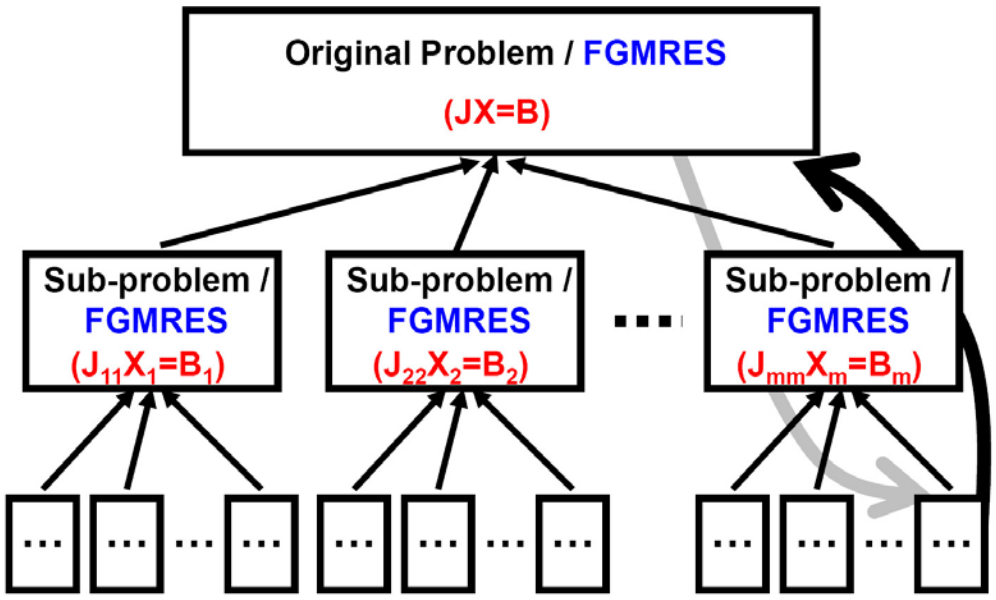

Fig. 35. The task-dependency graph of the hierarchical preconditioner.

Algorithm 1 PE allocation for hierarchical preconditioning

```
Inputs: a problem tree with root n; a set of P PEs;
        one problem is split into N sub-problems at the next level;
Allocate(n, P)
 1: Assign all PEs from $P$ to root node
 2: If $n$ does not have any child, return
 3: Else
 4:   Partition $P$ into $N$ non-overlapping subsets, $P^1, P^2, \cdots, P^N$:
 5:     IF $\lfloor \frac{P}{N} \rfloor == \frac{P}{N}$
 6:       $P^i$ has $P/N$ PEs $(1 \leq i \leq N)$
 7:     Elseif $(P > N)$
 8:       $P^i$ has $\lfloor \frac{P}{N} \rfloor + 1$ PEs $(1 \leq i < N)$ and
          $P^N$ has $P - (\lfloor \frac{P}{N} \rfloor + 1)(N - 1)$ PEs
 9:     Else
10:       $P^i$ has one PE $(1 \leq i \leq P)$ and others have no PE
11:   For each child $n_i$: Allocate($n_i$, $P^i$).
```

three PEs are simultaneously utilized for the computing work at the topmost level. From the second level downwards, a PE is assigned to solve a sub-matrix problem and its children. Similarly in the nine-PE case, the nine PEs are collectively used for the computing work at the topmost level. Since there are three sub-problems at the second level, three PE groups are formed, $\{P_1, P_2, P_3\}$, $\{P_4, P_5, P_6\}$ and $\{P_7, P_8, P_9\}$. Each group is assigned to a second-level subproblem and its third-level children.

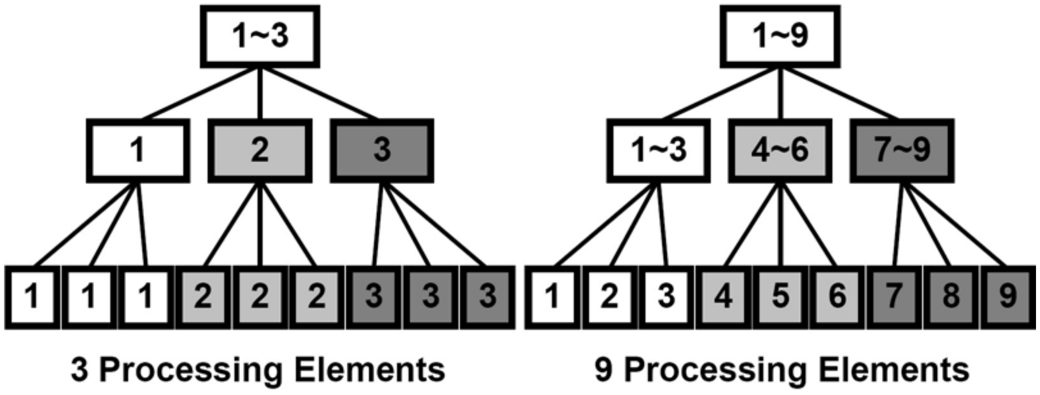

Fig. 36. Allocation of processing elements for hierarchical preconditioning.

A critical issue in the PE assignment is to prevent deadlock. A deadlock is a situation in which two or more dependent operations are waiting for each other to finish, which may occur in a variety of situations [78]. Let us consider Algorithm 1 in an MPI implementation, PEs P_1 and P_2 are assigned to solve the same-level matrix problems M_A and M_B in hierarchical preconditioning. And by the same algorithm, P_1 and P_2 may be also assigned to solve the sub-problems of M_A and M_B, respectively. But instead of the case above, if P_1 is assigned to solve a sub-problem of M_B and P_2 is assigned to solve a sub-problem of M_A, then a deadlock may happen. The two PEs have to send data to each other in order to proceed. When P_1 and P_2 simultaneously send the data and the system does not have enough buffer space for both, a deadlock may occur. It would be even worse if several pairs of such operations happen at the same time. The use of Algorithm 1 reduces the amount of inter-PE data transfer, therefore, avoids certain deadlock risks.

More generally, the PE allocation can be done while considering possible differences of the PE computing powers. In this case, the sizes of subproblems are matched to the computing powers of the assigned PEs. Such a size-dependent allocation algorithm is presented in Algorithm 2, where the cost of solving a linear matrix problem is assumed to be linearly proportional to the problem size.

Algorithm 2 Size-dependent PE allocation for hierarchical preconditioning

```
Inputs: a problem tree with root n; a set of P PEs; problem size S;
        one problem is split into N sub-problems at the next level;
        computing power weights of PEs : $w_1 \le w_2 \le \cdots \le w_P$
Allocate(n, P, S)
1: Assign all PEs to root node
2: If $n$ does not have any child, return
3: Else
4:    Partition $P$ into $N$ non-overlapping subsets: $P^1, P^2, \cdots, P^N$,
      with the total subset weights $w_{s,i}, (1 \le i \le N)$.
5:   Minimize the differences between $w_{s,i}$'s.
6:   Choose the size of the i-th child node $n_i$ as:
     $S_i = S \cdot w_{s,i} / \sum_{j=1}^{P} w_j$
7:    For each $n_i$: Allocate($n_i$, $P^i$, $S_i$ ).
```

Consider the example in Fig. 37, where each problem is recursively split to three sub-problems at the next level and the (sub)problems are denoted as $n_i, (1 \le i \le 13)$. Assume there are nine PEs with computing power weights $w_1 = 9$, $w_2 = 8$, $w_3 = 7$, $w_4 = 6$, $w_5 = 5$, $w_6 = 4$, $w_7 = 3$, $w_8 = 2$ and $w_9 = 1$, respectively. By using Algorithm **??**, all PEs ($P_1 \sim P_9$) to n_1 are assigned to solve the top-level problem. To provide the preconditioner for the top-level problem, the nine PEs are partitioned to three subsets to minimize the computing power differences among sub-problems n_2, n_3 and

n_4. For example, assign $\{P_1, P_6, P_7\}$ to n_2, $\{P_2, P_5, P_8\}$ to n_3, and $\{P_3, P_4, P_9\}$ to n_4, as shown in Fig. 37. The total computing power of all the PEs is 45 and those allocated to n_2, n_3 and n_4 are 16, 15 and 14, respectively. Therefore, if the size of the top-level problem is 180, the sizes for the second-level subproblems are 64, 60 and 56, respectively. Similarly, the sizes of the third-level subproblems and their PE allocations can be determined, as shown at the bottom of Fig. 37.

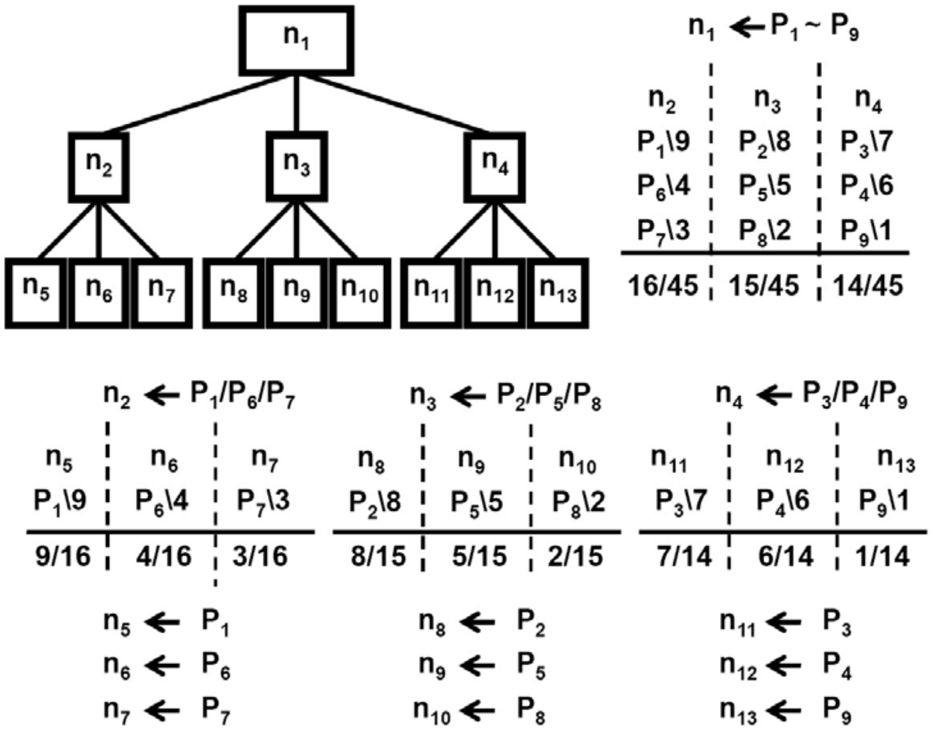

Fig. 37. Size-dependent PE allocation for a three-level preconditioner.

C. Extensions to parallel autonomous circuit and envelope-following analyses

the principal ideas of the parallel HB analysis are illustrated in the previous sections, mostly under the context of driven circuit simulation. The proposed parallel ideas can be further extended to two other HB-based analyses: autonomous circuit steady-state analysis and envelope-following analysis.

1. Parallel steady-state analysis of autonomous circuits

In an autonomous circuit regime, there are two problems not found with driven circuits. The period of the oscillator is unknown and must be determined, and the time origin is arbitrary and thus if one solution exists, then an infinite continuum of solutions exists. Therefore, HB analysis must be modified to handle autonomous circuits. To address the new issues for autonomous circuits (e.g. oscillators), a modified HB analysis has been applied to oscillator simulation by adding the fundamental frequency to the list of unknowns and an equation to enforce the constraint that solutions be isolated from one other [60]. However, autonomous circuit simulation using this method has proven to be difficult due to a small region of convergence and the existence of degenerate DC solution. Therefore,careful implementation and special techniques are have been developed [79–82].

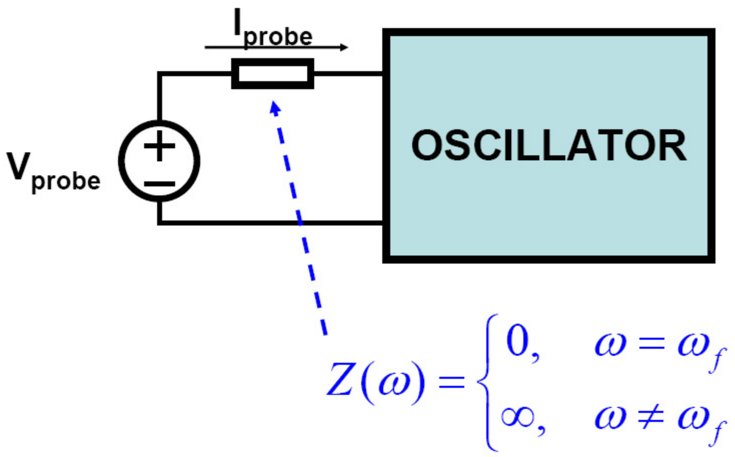
I_{probe}
V_{probe}
OSCILLATOR
$Z(\omega) = \begin{cases} 0, & \omega = \omega_f \\ \infty, & \omega \neq \omega_f \end{cases}$

Fig. 38. Voltage probe.

In [79], a two-tier approach for autonomous circuit HB simulation has been proposed. In this approach, the concept of voltage probe as shown in Fig. 38 is introduced to transform the original autonomous circuit problem to a set of closely-related driven circuit problems so that the original problem can be solved more efficiently. As shown

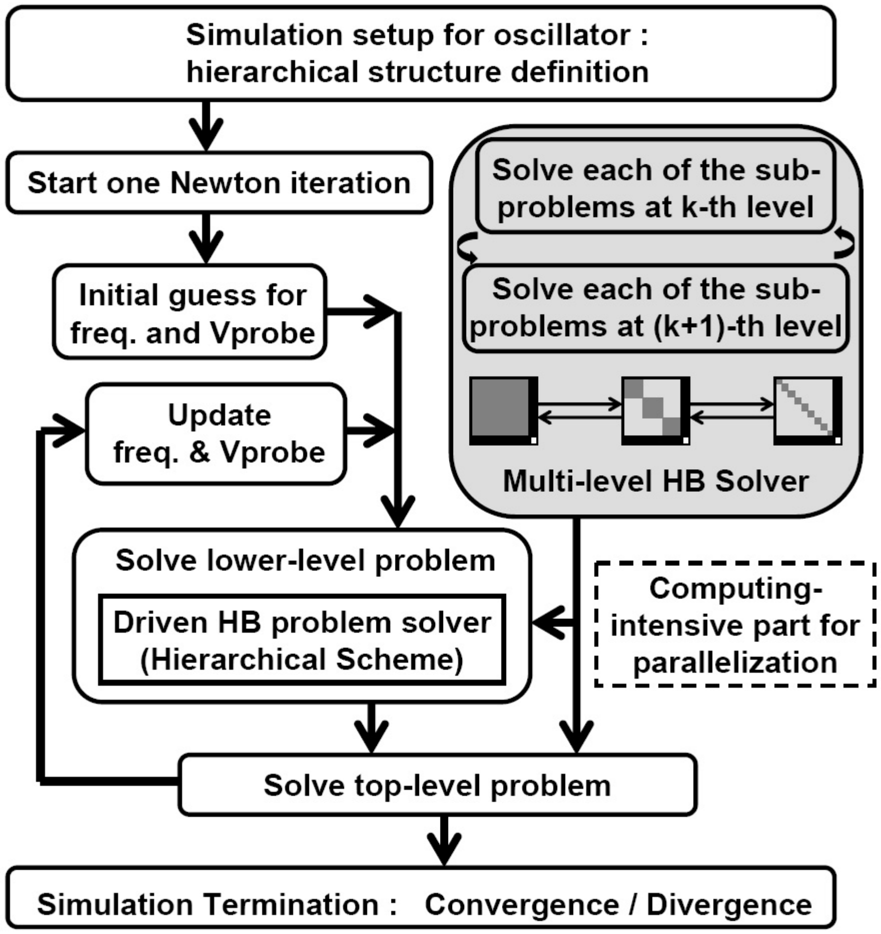
Simulation setup for oscillator : hierarchical structure definition
Start one Newton iteration
Initial guess for freq. and Vprobe
Update freq. & Vprobe
Solve each of the sub-problems at k-th level
Solve each of the sub-problems at (k+1)-th level
Multi-level HB Solver
Solve lower-level problem
Driven HB problem solver (Hierarchical Scheme)
Computing-intensive part for parallelization
Solve top-level problem
Simulation Termination : Convergence / Divergence

Fig. 39. Parallelizable autonomous circuit HB analysis.

in Fig. 39, based on some initial guesses of the probe voltage and the steady-state frequency, a driven-circuit-like HB problem at the second level (the lower tier) is formulated in a form

$$\begin{bmatrix} J & e_m^c(1) & e_m^s(1) \\ e_m^c(1)^T & 0 & 0 \\ e_m^s(1)^T & 0 & 0 \end{bmatrix} \begin{bmatrix} X^{(j+1)} \\ I_{probe}^{c(j+1)} \\ I_{probe}^{s(j+1)} \end{bmatrix} = \begin{bmatrix} F^{(j)} \\ V_{probe} \\ 0 \end{bmatrix}, \tag{6.8}$$

where $e_m^c(1)$ *and* $e_m^s(1)$ are the unit vectors that select the cosine and sine parts of the fundamental frequency of the probing node. I_{probe}^c *and* I_{probe}^s are the cosine and sine parts of the probe current. F is the right-hand-side vector of the circuit. j means the j^{th} Newton iteration. After solving the problem at the second level, the obtained probe current I_{probe} is used to update the probe voltage and the steady-state frequency at the top level (the upper tier) by solving a two-dimensional nonlinear problem

$$\begin{cases} \Re\left(I_{probe}\left(V_{probe}, \omega_{osc}\right)\right) = 0 \\ \Im\left(I_{probe}\left(V_{probe}, \omega_{osc}\right)\right) = 0 \end{cases}, \tag{6.9}$$

where $\Re(\cdot)$ and $\Im(\cdot)$ take the real part and imaginary part. To solve the linearized equation for top level problem at each Newton iteration, the following Jacobian matrix is computed

$$J_{probe} = \begin{bmatrix} \frac{\partial \Re\left[I_{probe}\right]}{\partial V_{probe}} & \frac{\partial \Re\left[I_{probe}\right]}{\partial \omega} \\ \frac{\partial \Im\left[I_{probe}\right]}{\partial V_{probe}} & \frac{\partial \Im\left[I_{probe}\right]}{\partial \omega} \end{bmatrix}. \tag{6.10}$$

Once the updated probe voltage V_{probe} and frequency ω are obtained at the top level, a new driven-circuit-like HB problem at the second level is formulated and solved. The process repeats until the probe current comes to (approximately) zero.

Since solving the second-level HB problem dominates the overall computational complexity, it becomes the main target for parallelization. The linearized HB problem (6.8) at the lower tier can be represented in a general form

$$\begin{bmatrix} A_{nN\times nN} & B_{nN\times l} \\ C_{l\times nN} & D_{l\times l} \end{bmatrix} \cdot X_{(nN+l)\times 1} = V_{(nN+l)\times 1}, \tag{6.11}$$

where n and N are the numbers of the circuit unknowns and harmonics, respectively, and $l(l << nN)$ is the number of the additionally appended variables corresponding to the steady-state frequency and the probe voltage. It is not difficult to see that the structure of matrix block $A_{nN \times nN}$ is identical to the Jacobian matrix in a driven circuit HB analysis. To see how the parallelization ideas for driven circuits can be extended for autonomous circuits, (6.11) is rewritten in the following partitioned form (6.12) with neglecting matrix subscripts

$$\begin{cases} AX_1 + BX_2 = V_1 \\ CX_1 + DX_2 = V_2 \end{cases} . \tag{6.12}$$

Utilizing the first equation in (6.12), X_1 can be expressed in terms of X_2 as:

$$X_1 = A^{-1}(V_1 - BX_2). \tag{6.13}$$

Then substituting (6.13) into the second equation in (6.12) leads to

$$X_2 = (D - CA^{-1}B)^{-1}(V_2 - CA^{-1}V_1). \tag{6.14}$$

The dominant computational cost for getting X_2 comes from solving the two linearized matrix problems associated with $A^{-1}B$ and $A^{-1}V_1$. When X_2 is available, X_1 can be obtained by solving the third linearized matrix problem defined by A as shown in (6.13). This overall procedure is illustrated in Fig. 40. All the three matrix problems are defined by matrix A, which has a structure identical to the Jacobian of a driven circuit. As a result, the same parallel preconditioning technique described before can be applied [49].

2. Parallel envelope-following analysis

To analyze the periodic or quasi-periodic circuit responses with slowly varying amplitudes, envelope-following analysis has been introduced [83–88]. The principal idea of the HB-based envelope-following analysis is to handle the slowly varying amplitude, called envelope, of the fast carrier separately from the carrier itself, as shown in

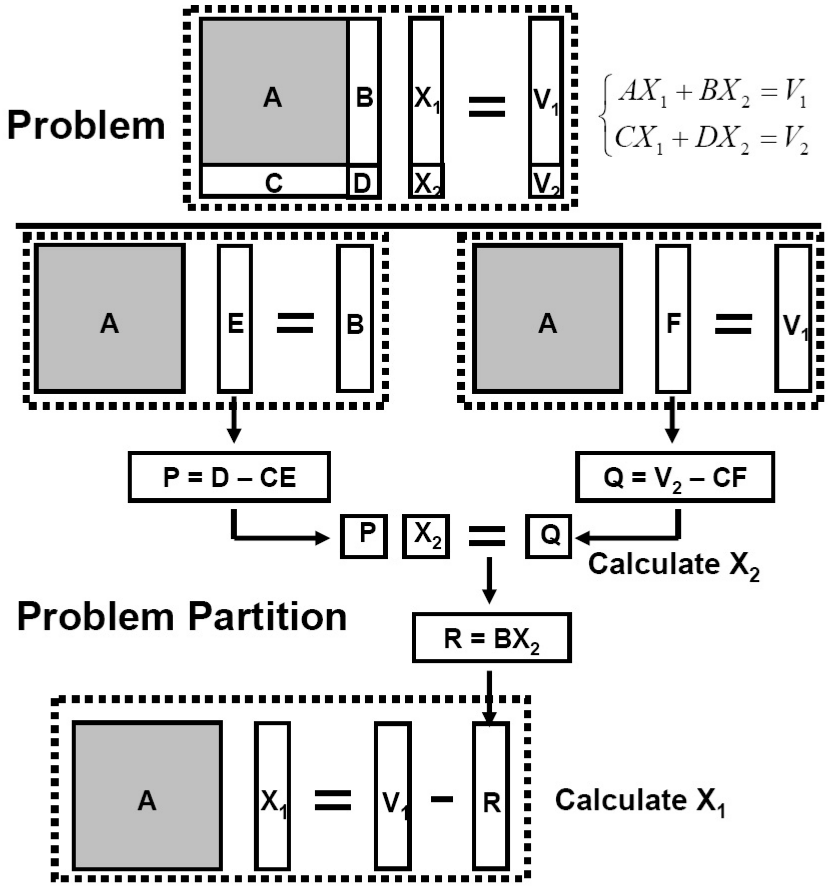

Fig. 40. Partitioning of the Jacobian of autonomous circuits.

Fig. 41 [86–88]. The signals in the envelope-following analysis are non-periodic and

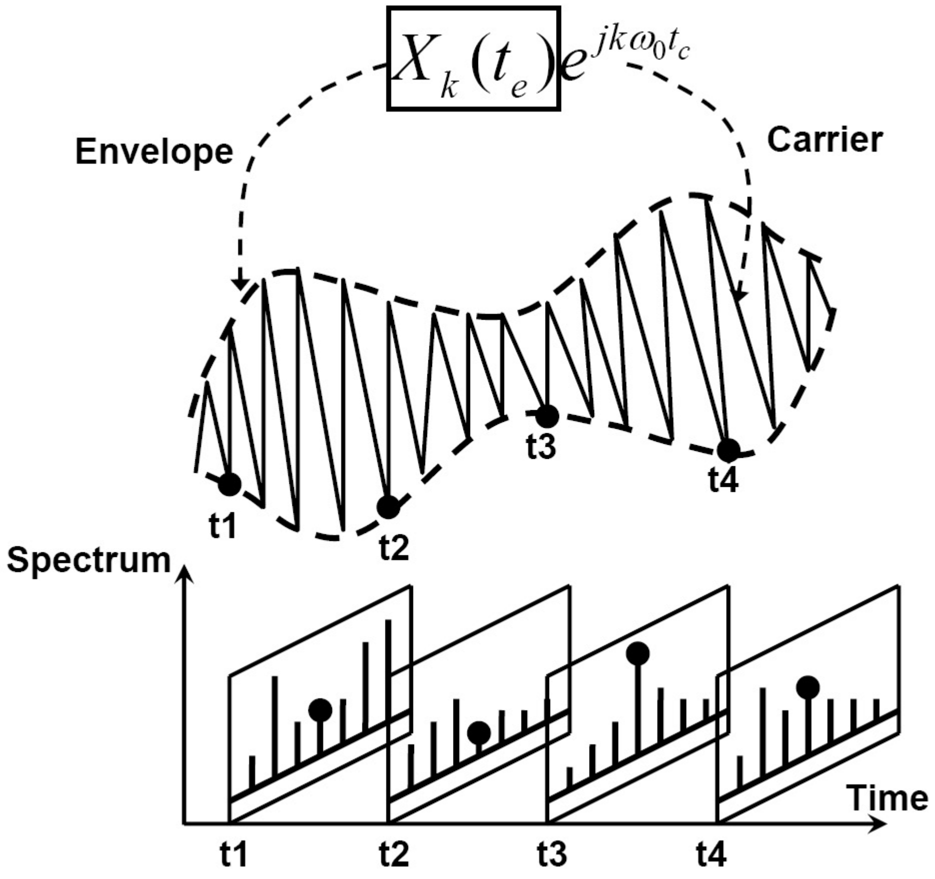

Fig. 41. Envelope-following analysis.

can be expressed using Fourier series expansions as

$$x(t) = \sum_{k=-K}^{K} X_k(t)e^{jk\omega_0 t}, N = 2K + 1, \tag{6.15}$$

where $X_k(t)$ is assumed to vary slowly with respect to the period of the carrier $T_0 = 2\pi/\omega_0$. Accordingly, the general circuit equations in (**??**) can be expressed as

$$\begin{gathered} h(t) = h(t_e, t_c) = \sum_{k=-K}^{K} [jk\omega_0 Q_k(t_e) \\ + \tfrac{d}{dt} Q_k(t_e) + G_k(t_e) - U_k(t_e)]e^{jk\omega_0 t_c} \end{gathered}, \tag{6.16}$$

where different time variables t_e, t_c are used for the envelope and the carrier. Correspondingly, the Fourier coefficients shall satisfy the following equations

$$\begin{aligned} H(X(t_e)) &= \Omega\Gamma q(\cdot)\Gamma^{-1}X(t_e) + \frac{d}{dt_e}\Gamma q(\cdot)\Gamma^{-1}X(t_e) \\ &\quad +\Gamma f(\cdot)\Gamma^{-1}X(t_e) - U(t_e) = 0, \end{aligned} \tag{6.17}$$

which can be solved by using a numerical integration method. Applying Backward Euler (BE) to discretize (6.17) over a set of time points $(t_1,\ t_2,\ \cdots,\ t_q,\ \cdots)$ leads to

$$\begin{aligned} &\left(\Gamma q(\cdot)\Gamma^{-1}X(t_q) - \Gamma q(\cdot)\Gamma^{-1}X(t_{q-1})\right)/(t_q - t_{q-1}) \\ &+\Omega\Gamma q(\cdot)\Gamma^{-1}X(t_q) + \Gamma f(\cdot)\Gamma^{-1}X(t_q) - U(t_q) = 0. \end{aligned} \tag{6.18}$$

To solve this nonlinear problem using the Newton's method, the Jacobian is needed

$$\begin{aligned} &J_{env} = \tfrac{\Gamma C\Gamma^{-1}}{t_q-t_{q-1}} + \Omega\Gamma C\Gamma^{-1} + \Gamma G\Gamma^{-1} = \\ &\begin{bmatrix} \Omega_1 + \frac{I_1}{t_q-t_{q-1}} & & \\ & \ddots & \\ & & \Omega_m + \frac{I_m}{t_q-t_{q-1}} \end{bmatrix} \cdot C_c + G_c, \end{aligned} \tag{6.19}$$

where the equation is partitioned into m blocks in a way similar to (6.3); $I_1,\ I_2,\ \cdots,\ I_m$ are identity matrices with the same dimensions as the matrices $\Omega_1,\ \Omega_2,\ \cdots,\ \Omega_m$, respectively; Circulants C_c and G_c have the same forms as in (6.4). Similar to the treatment taken in (6.6), a parallel preconditioner can be formed by discarding the off-block diagonal entries of (6.4), which leads to m decoupled linear problems of smaller dimensions

$$\left\{\begin{array}{c} [(\Omega_1 + \frac{I_1}{(t_q - t_{q-1})})C_{c11} + G_{c11}]X_1 = B_1 \\ [(\Omega_2 + \frac{I_2}{(t_q - t_{q-1})})C_{c22} + G_{c22}]X_2 = B_2 \\ \vdots \\ [(\Omega_m + \frac{I_m}{(t_q - t_{q-1})})C_{cmm} + G_{cmm}]X_m = B_m \end{array}\right. . \tag{6.20}$$

The mathematical structures of these sub-problems are identical to those in the standard HB, hence they can be implicitly formed in the same way. The decomposition above can be extended hierarchically, giving rise to a hierarchical preconditioner. The algorithm flow of such a parallel preconditioned envelope-following analysis is shown in Fig. 42.

D. Implementation issues

The proposed parallel simulation approach is implemented using MPI on distributed computing platform (e.g. a cluster of workstations). And for comparisons, the proposed parallel HB technique has also been implemented on the shared-memory platform. As it is known, the main runtime overheads on the distributed platform come from the inter-PE communications in the network. For example, for parallel device model evaluations, the different PEs correspond to the evaluations for different devices. The evaluation results should be collected together and then be assigned to different PEs through the network for the following parallel linearized problem solving. In the proposed parallel HB simulator, other parallel operations also require the similar inter-PE communications. Therefore, one main implementation issue is to reduce the communication overheads among the networked workstations. For this purpose, non-blocking MPI routines are adopted instead of blocking ones to overlap computation and communication. In blocking operations, the overhead for guaran-

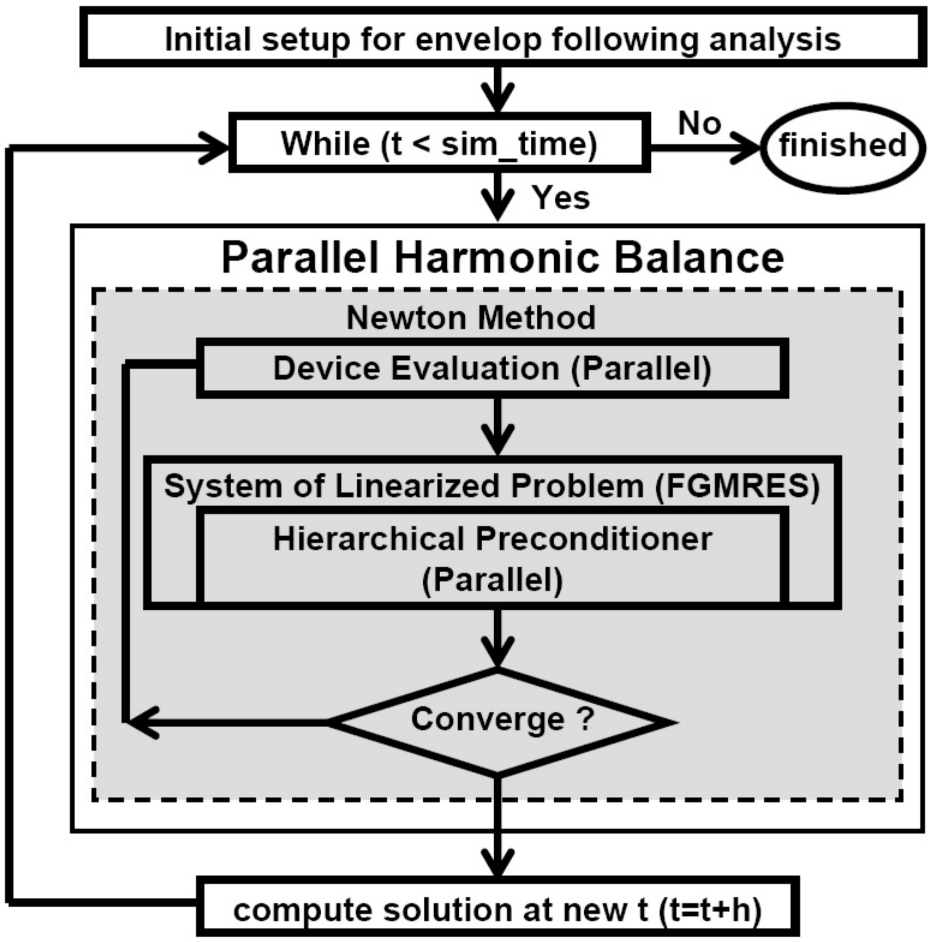
Initial setup for envelop following analysis
While (t < sim_time)
No
finished
Yes
Parallel Harmonic Balance
Newton Method
Device Evaluation (Parallel)
System of Linearized Problem (FGMRES)
Hierarchical Preconditioner (Parallel)
Converge ?
compute solution at new t (t=t+h)

Fig. 42. The algorithm flow of parallel envelope-following analysis.

teeing semantic correctness is paid in the form of idling/buffer management. On the other hand, non-blocking operations are useful for performance optimization via the reduction of communication overhead.

Consider the example in Fig. 37. The solutions of subproblems n_5, n_6 and n_7 computed by PEs P_1, P_6 and P_7, respectively, need to be all sent to one PE, say P_1, which also works on a higher-level parent problem. Since multiple sub-problems are being solved concurrently, P_1 may not immediately respond to the request from P_6 (or P_7), especially when the amount of sending data is large. If blocking operations are used, it is expected that the communication cost will be high. However, when non-blocking operations are adopted, the same time interval can be used to perform any computation that does not depend upon the data being sent. A useful idea, as shown in Fig. 43, is to split the data into several segments. At a time, P_6 (or P_7) only prepares one segment of data and sends a request to P_1. Then, the PE can prepare the next segment of data to be sent. As such, the communication and computation can be partially overlapped.

It deserves mentioning that the emergence of the multicore platform provides a new opportunity for parallel computing. By taking the advantage of the shared-memory hierarchy, the inter-PE communications may be reduced. But for large circuit simulations, the limited shared-memory resources must be carefully handled.

E. Experimental results

The proposed approach is implemented in C/C++ with the MPICH library [53] used for parallel processing. The FFTW package is used for FFT/IFFT operations [89] and the FGMRES solver is provided through the PETSC package [90]. The experiments are conducted on a network of Linux machines with single or dual-core processors.

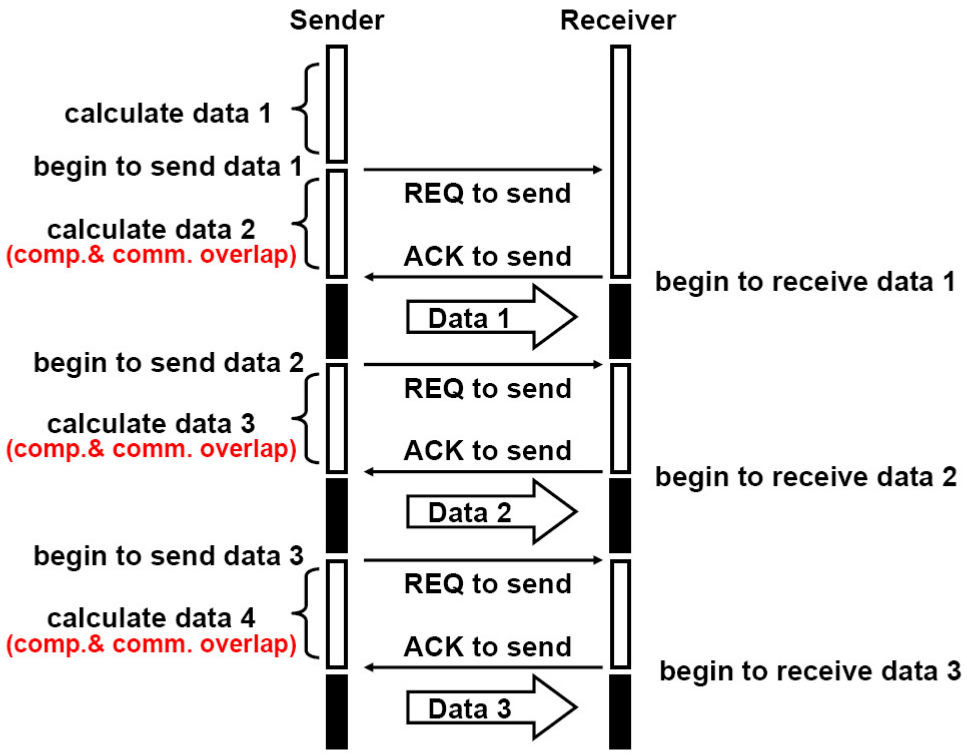

Fig. 43. Non-blocking data transfers.

The total number of CPU cores is nine.

1. Performance of driven circuit simulation

A set of testing driven circuits listed in Table IX are used to demonstrate the performance of the parallel HB simulations. To make fair comparisons between the serial and parallel HB simulations, the same convergence tolerances and hierarchal preconditioner structure are employed. When using the hierarchical preconditioning technique for HB simulation, a trisection three-level hierarchy is used, where the size of each sub-problem is one third of that of its parent problem.

Table IX. Descriptions of the driven circuits.

Index	Description of circuits	Nodes	Freqs	Unknowns
1	frequency divider	17	100	3,383
2	DC-DC converter	8	150	2,392
3	diode rectifier	5	200	1,995
4	double-balanced mixer	27	188	10,125
5	low noise amplifier	43	61	5,203
6	LNA + mixer	69	86	11,799
7	RLC mesh circuit	1,735	10	32,965
8	digital counter	86	50	8,514

As a reference, the runtime information of the serial HB simulations with the BD preconditioner [61] and the hierarchical preconditioner is shown in Table X. The simulation results for the former preconditioner cases are shown in the $2nd$, $3rd$ and $4th$ columns, where *N-Its* and *K-Its* indicate the total numbers of Newton and FGMRES iterations required to reach the convergence during the entire simulation. $T(s)$ records the CPU times in seconds. The results for the latter preconditioner cases are shown in the $5th$, $6th$ and $7th$ columns, where *K-Its* indicates the total number of top-level FGMRES iterations.

First, in order to give the insights of the runtime speedup contributions from the different parallel parts in the HB simulation, the individual runtime percentage of three key steps : device model evaluation, hierarchical preconditioning and matrix-

Table X. Statistics of the serial HB simulations for the driven circuits.

Index	Serial BD			Serial Hierarchical		
	N-Its	K-Its	T(s)	N-Its	K-Its	T(s)
1	13	5,187	478	15	1,996	229
2	50	5,331	904	47	1,036	188
3	13	1,931	228	14	353	49
4	27	981	79	26	159	24
5	40	5,303	1,532	41	578	185
6	24	1,201	195	23	299	58
7	36	4,726	286	34	1,013	69
8	77	8,342	3,127	75	2,308	1,310

vector product in serial HB simulation, and also the runtime speedups when each of them is parallelized on a 3-CPU network are listed for the LNA-mixer circuit and the RLC mesh circuit. The results are shown in Table XI, in which the columns below 'T(s)' indicate the runtimes of the parallel simulations; the columns below '%' correspond the percentages of the runtime contributions in the serial simulations; and the columns below 'X' show the speedups obtained by parallelization. In the remaining part of Section E, all the parallel simulation results are obtained by simultaneously parallelizing all of the parallelizable parts in HB simulation.

Table XI. Runtime statistics of three key steps and their parallelization on the 3-CPU platform.

Circuit	Device Evaluation			Preconditioning			Matrix-vector Product		
	T(s)	%	X	T(s)	%	X	T(s)	%	X
LNA+mixer	45	38	1.29	49	33	1.18	50	29	1.14
RLC mesh	62	20	1.11	53	39	1.29	59	33	1.17

Next, the parallel HB analyses using the BD and hierarchical preconditioner are compared with their serial counterparts, respectively. The results obtained on the 3-CPU and 9-CPU platforms are shown in Tables XII. In the table, the columns below 'T1(s)', 'T3(s)' and 'T2(s)', 'T4(s)' correspond to the runtimes of the parallel HB simulations with the BD preconditioner and those with the hierarchical preconditioner, respectively. The columns below 'X1'-'X4' indicate the runtime speedups over their serial counterparts, respectively.

On the 3-CPU platform, the average speedup values below the columns 'X1' and

Table XII. Statistics of the parallel HB simulations on the 3-CPU / 9-CPU platforms for the driven circuits.

Index	Parallel 3-CPU Platform				Parallel 9-CPU Platform			
	BD		Hierarchical		BD		Hierarchical	
	T1(s)	X1	T2(s)	X2	T3(s)	X3	T4(s)	X4
1	254	1.88	125	1.83	120	3.98	61	3.79
2	478	1.89	103	1.83	229	3.95	49	3.81
3	125	1.82	28	1.77	62	3.69	14	3.54
4	44	1.78	14	1.68	20	3.94	6	3.76
5	786	1.95	100	1.85	409	3.75	52	3.53
6	112	1.74	34	1.68	53	3.67	16	3.56
7	154	1.85	38	1.80	76	3.76	19	3.62
8	1,587	1.97	686	1.91	786	3.98	341	3.84

'X2' are 1.86x, 1.79x, respectively; On the 9-CPU platform, these average runtime speedups are 3.84x, 3.68x, respectively. The advantages of the parallel hierarchical preconditioner over the parallel BD preconditioner can be clearly seen as well. This is the reason why the former is preferred. The runtime speedups of the parallel hierarchical preconditioner over its serial counterpart as a function of the number of processors for three test circuits are shown in Fig. 44.

Although in this work, the parallel implementation is mainly focused on the distributed-memory platform. The proposed parallel method can also be implemented on the shared-memory platform. It is interesting to compare the simulation results on the shared-memory platform and those on the distributed-memory platform. It deserves mentioning that the same FGMRES algorithm has been implemented for both the shared-memory and distributed-memory platforms for fair comparisons. In Fig. 45, the experimental results of the frequency-divider and the DC-DC are shown. It can be observed that the runtime speedups for both the MPI implementation and the Pthreads implementation are similar in the experiments. But it can be expected that with the trend of more processing cores being integrated on one chip, the communication overheads among the distributed workstations would be more significant than those on the shared-memory platform. As a result, the shared-memory platform is more promising under the conditions of high inter-PE communications.

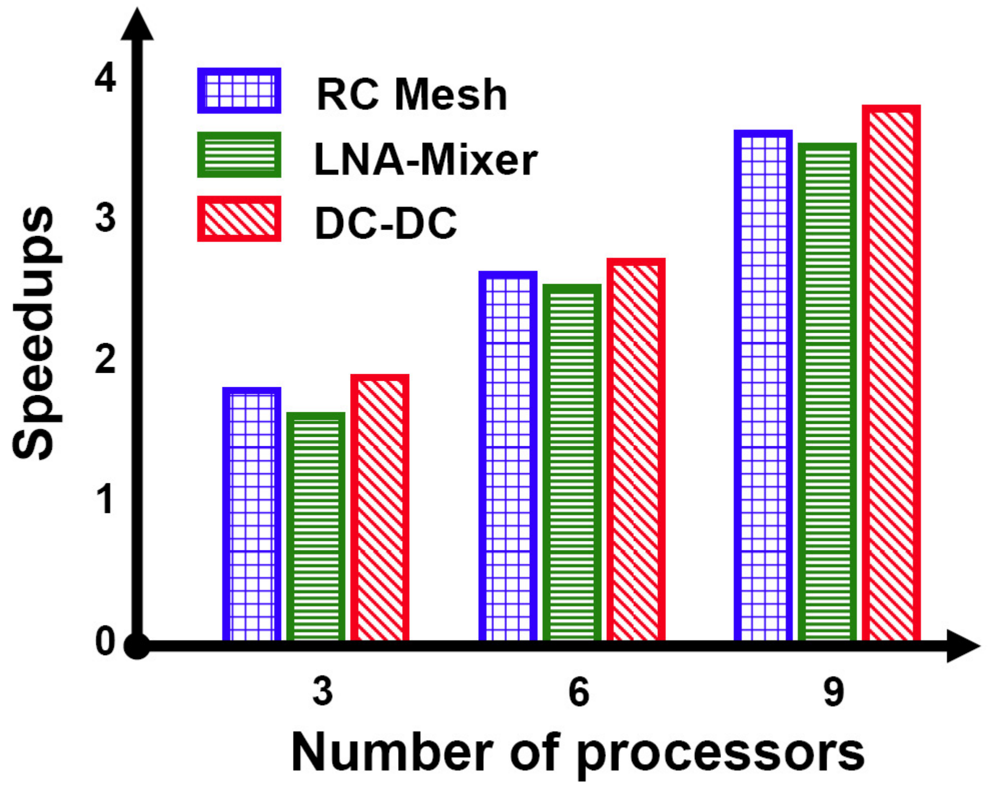

Fig. 44. The runtime speedups of the parallel HB with hierarchical preconditioning vs. the number of the processors.

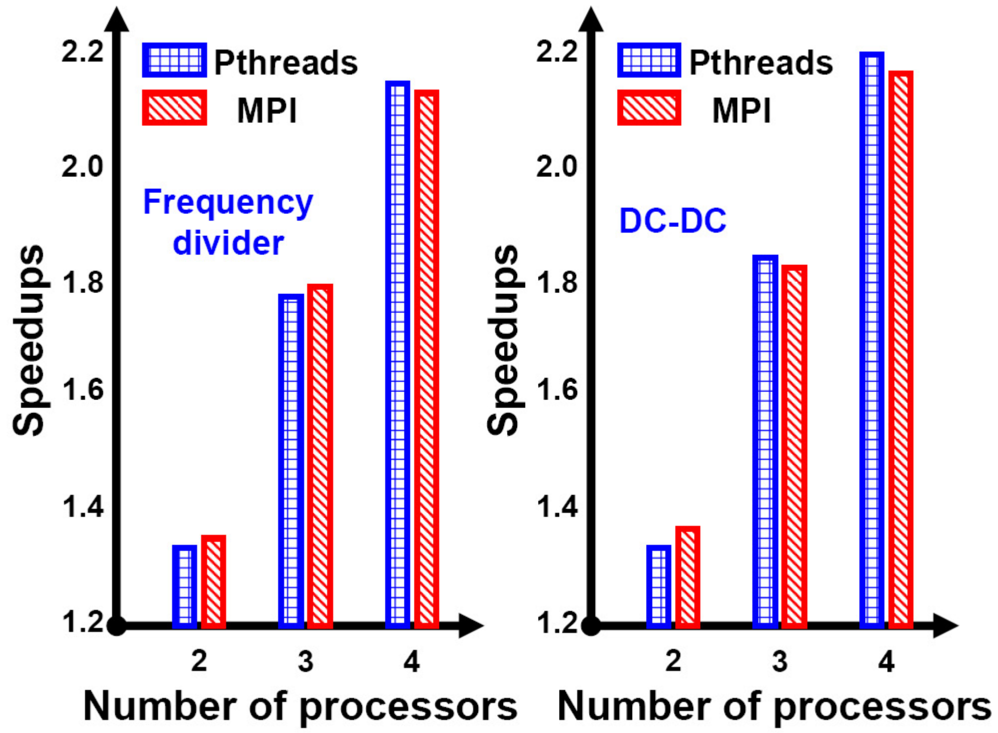

Fig. 45. Comparison of shared-memory and distributed-memory implementations.

2. Performance of autonomous circuit simulation

A set of oscillators described in Table XIII are used to verify the proposed parallel steady-state analysis for autonomous circuits.

Table XIII. Descriptions of the autonomous circuits.

Description of circuits	Nodes	Freqs	Unknowns
11 stages ring oscillator	13	50	1,289
13 stages ring oscillator	15	25	737
15 stages ring oscillator	17	20	665
LC oscillator	12	30	710
digital-controlled oscillator	152	10	2890

Two versions of the two-tier method [79] are implemented, one with the BD preconditioner and the other with the hierarchical preconditioner. The runtimes of the serial implementations of the two versions are listed in the columns labeled as "Serial Platform" in Table XIV. At the same time, the runtimes of the parallel simulations with the BD and hierarchical preconditioners on the 3-CPU and 9-CPU platforms are also shown in Table XV. The columns below 'X3' and 'X5' are the speedups of parallel simulations with the BD preconditioners. And the columns below 'X4' and 'X6' are the speedups of parallel simulations with the hierarchical preconditioners.

Table XIV. Statistics of the HB simulations on serial platform for the oscillators.

Circuit	Serial Platform			
	Two-tier BD		Two-tier Hier.	
	T1(s)	N-Its	T2(s)	N-Its
11 stages ring oscillator	162	50	87	44
13 stages ring oscillator	122	31	64	28
15 stages ring oscillator	108	28	56	24
LC oscillator	141	43	75	38
digital-controlled oscillator	1233	41	680	39

On the 3-CPU platform, the average values below the columns 'X3' and 'X4' are 1.73x, 1.70x, respectively; On the 9-CPU platform, these average values are 3.90x and 3.79x respectively. It can be observed that the proposed parallel method brings favorable speedups over both its serial implementation and the parallel counterpart with BD preconditioner.

Table XV. Statistics of the HB simulations on parallel platforms for the oscillators.

Circuit	Parallel 3-CPU Platform				Parallel 9-CPU Platform			
	Two-tier BD		Two-tier Hier.		Two-tier BD		Two-tier Hier.	
	T3(s)	X3	T4(s)	X4	T5(s)	X5	T6(s)	X6
11 stages ring oscillator	94	1.72	52	1.69	41	3.98	23	3.84
13 stages ring oscillator	70	1.74	37	1.72	31	3.96	17	3.86
15 stages ring oscillator	62	1.74	33	1.70	28	3.79	15	3.67
LC oscillator	83	1.69	45	1.67	37	3.81	20	3.70
digital-controlled oscillator	670	1.77	391	1.74	311	3.96	176	3.87

3. Performance of envelope-following simulation

The proposed parallel technique can be extended to the HB based envelope-following simulation. To demonstrate the proposed parallel envelope-following analysis, the theoretical analysis for the amplitude modulation has been given in the previous section. Correspondingly, two test circuits (a power amplifier and a double-balanced mixer) involving amplitude modulation are used to validate the proposed method, as shown in Fig. 46 and Fig. 47.

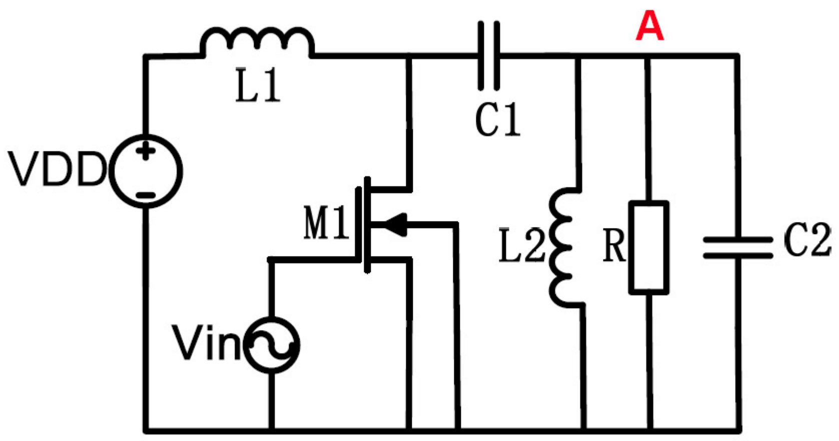

Fig. 46. A schematic of power amplifier.

The experimental results for both the transient simulations and the envelope-following simulations are shown. For the power amplifier case, the frequencies of the carrier and the modulating signal are 1MHz and 1KHz, respectively. The time step

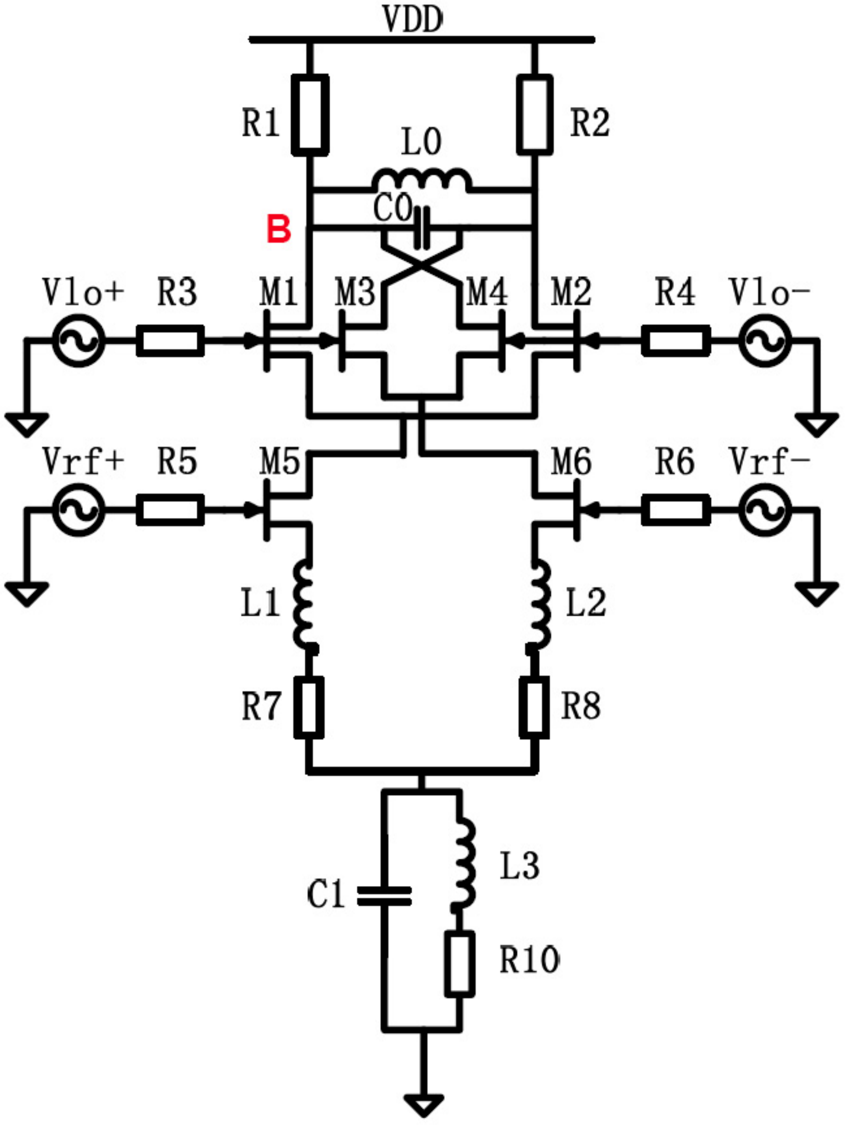

Fig. 47. A schematic of double-balanced mixer.

for the envelope-following simulation is 50 times of the period of the carrier. The waveforms of the transient simulation and the envelope-following simulation at node 'A' are shown in Fig. 48 and Fig. 49. Two different envelopes for the two different time shifts are plotted in Fig. 49. For the mixer case, the frequencies of the carrier

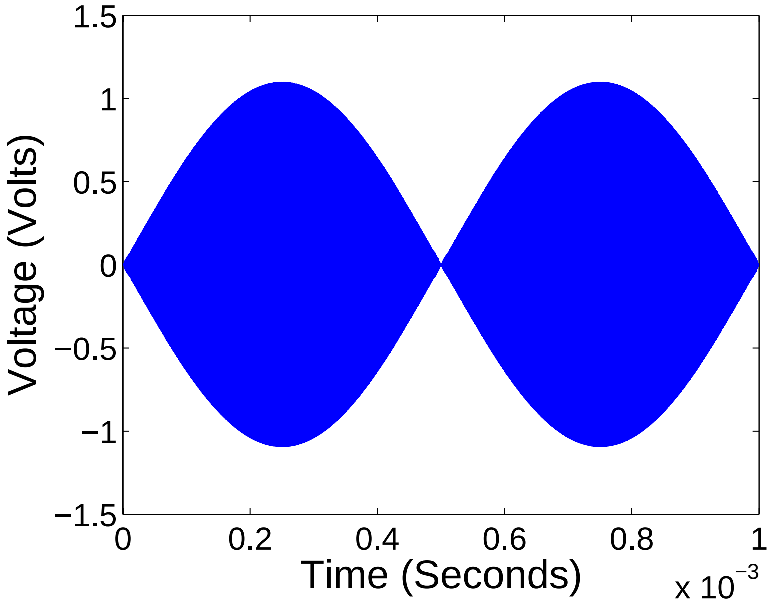

Fig. 48. Transient simulation of the power amplifier.

and the modulating signal are 2GHz and 10MHz, respectively. And the time step is 5 ns for the envelope-following simulation. The waveforms of the transient simulation and the envelope-following simulation at node 'B' are shown in Fig. 50 and Fig. 51 respectively. In the figure of the envelope-following simulation, four envelopes for the four different time shifts are plotted.

In Table XVI and Table XVII, the runtime statistics of the envelope-following simulations is listed. As a reference, the runtimes of the serial transient simulation, the serial envelope-following simulations with the BD and the hierarchical preconditioners are listed in the columns below "Serial Platform" in the table. And the columns

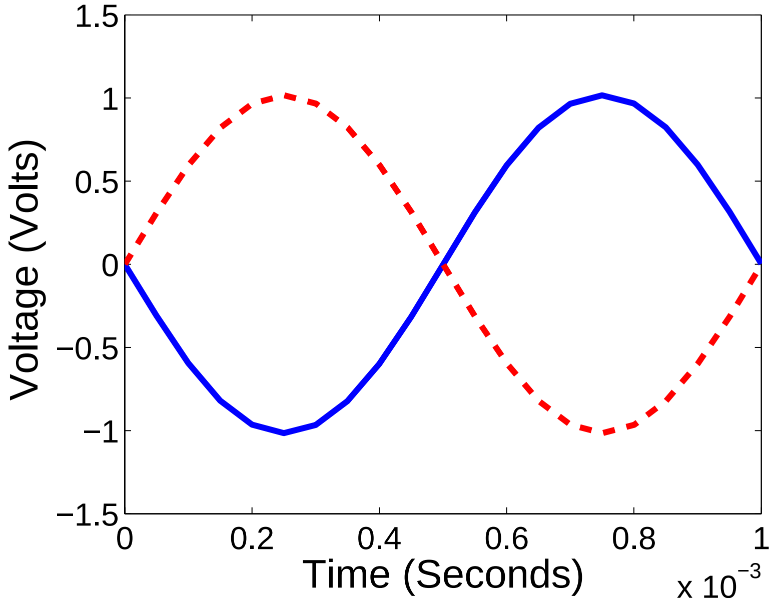

Fig. 49. Envelope-following simulation of the power amplifier.

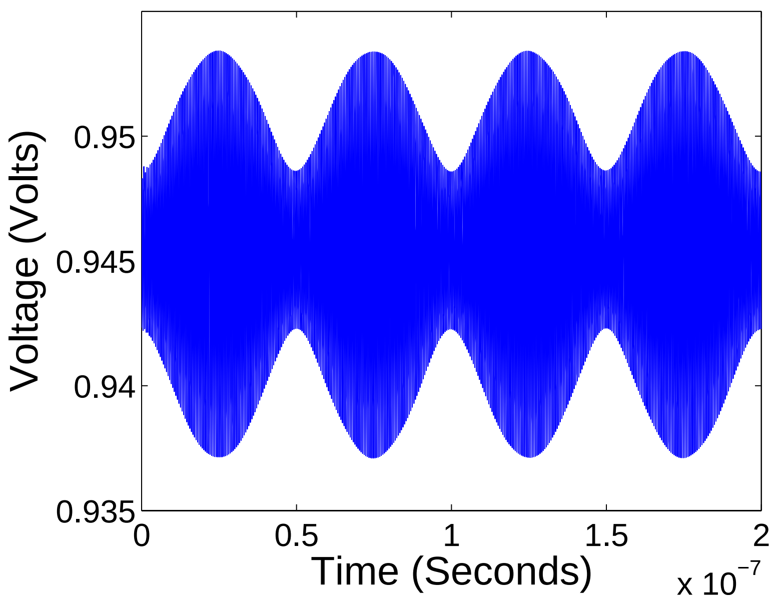

Fig. 50. Transient simulation of the double-balanced mixer.

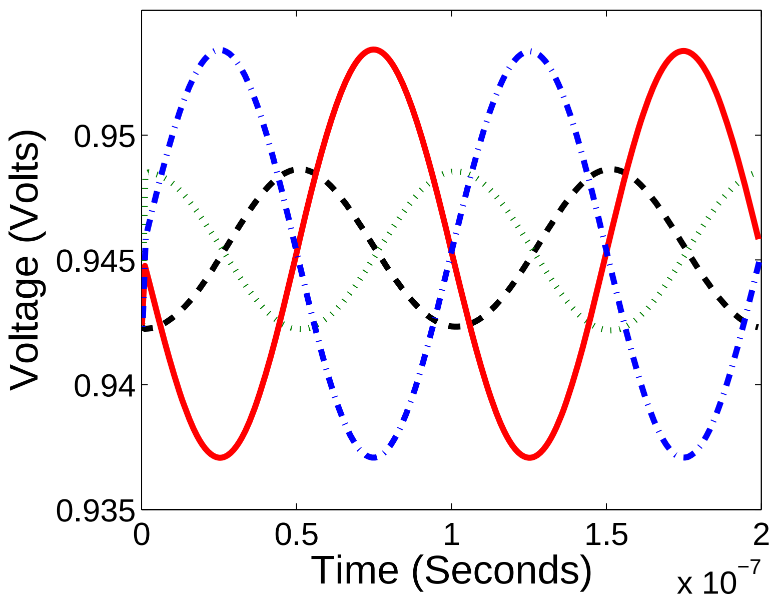

Fig. 51. Envelope-following simulation of the double-balanced mixer.

below 'X2' and 'X3' indicate the speedups of the envelope-following simulation over the transient simulation. In the columns labeled as "Parallel 3-CPU Platform" and "Parallel 9-CPU platform", the experimental results of the parallel envelope-following simulations with the BD preconditioner and the hierarchical preconditioner on the three and nine CPUs are shown. The columns below 'X4'-'X7' indicate the runtime speedups of the parallel envelope-following analyses over their serial counterparts. The runtime benefits of the proposed parallel approach are clearly seen.

Table XVI. Statistics of the envelope-following simulations on serial platform.

Circuit	Serial Platform				
	Transient	BD		Hierarchical	
	T1(s)	T2(s)	X2	T3(s)	X3
Power Amplifier	1,115	100	11.2	35	32.3
Double-balanced Mixer	1,776	131	13.6	51	34.9

Table XVII. Statistics of the envelope-following simulations on parallel platforms.

Circuit	Parallel 3-CPU Platform				Parallel 9-CPU Platform			
	BD		Hierarchical		BD		Hierarchical	
	T4(s)	X4	T5(s)	X5	T6	X6	T7	X7
Power Amplifier	57	1.74	21	1.66	25	4.02	9	3.74
Double-balanced Mixer	76	1.72	31	1.65	33	3.96	14	3.70

F. Summary

In this chapter, a parallel HB simulation framework is developed, which is built upon a parallelizable hierarchical preconditioning technique. By parallelizing the dominant computational portions of HB analysis and reducing the communication overhead through careful implementation, the proposed parallel approach has been successfully applied to different types of HB-based analyses. The experimental results have shown favorable runtime performances of the proposed approach for not only the steady-state simulation of driven and autonomous circuits, but also the HB-based envelope-following analysis.

CHAPTER VII

APPLICATION OF PARALLEL HARMONIC BALANCE SIMULATION TO MASSIVE CLOCK MESHES

A. Introduction

High performance IC designs impose stringent design specifications on clock distribution networks, where clock skews must be well controlled even under the presence of environmental and process variations. As a result, clock meshes are gaining increasing popularity due to their inherent low skew and immunity to variations. While clock meshes are often analyzed in time-domain for the purpose of verification as well as tuning, the massive couplings within the passive mesh structure and in between a large number of clock drivers are challenging to handle. In contrast, frequency-domain steady-state simulation techniques such as HB analysis are specifically advantageous since the massive passive mesh structure can be rather compactly represented using matrix transfer function matrices at a discrete set of harmonic frequencies. The remaining challenge, however, is to develop HB techniques that can efficiently simulate highly nonlinear steady-state problems corresponding to a large number of tightly coupled clock drivers. In this chapter, the proposed parallel HB simulation technique is employed to solve massive clock meshes problem to efficiently improve the runtime performance.

Fig. 52 illustrates a non-tree clock distribution network topology commonly used in high performance microprocessor designs [91, 92]. A standard H-tree is employed to distribute the clock signals at the top levels of clock distribution network while a mesh that is spanning the complete chip drives the bottom level clock drivers or flip-flops. From a network analysis point of view, the mesh structure is particularly problematic. A complete full-chip mesh model encompassing various capacitive and inductive coupling effects can be fairly complex, e.g., it may reach a complexity of a few million circuit variables. Furthermore, such large passive mesh structure may tightly couple with a large number (e.g. tens or hundreds) of mesh clock drivers, presenting a daunting circuit simulation task.

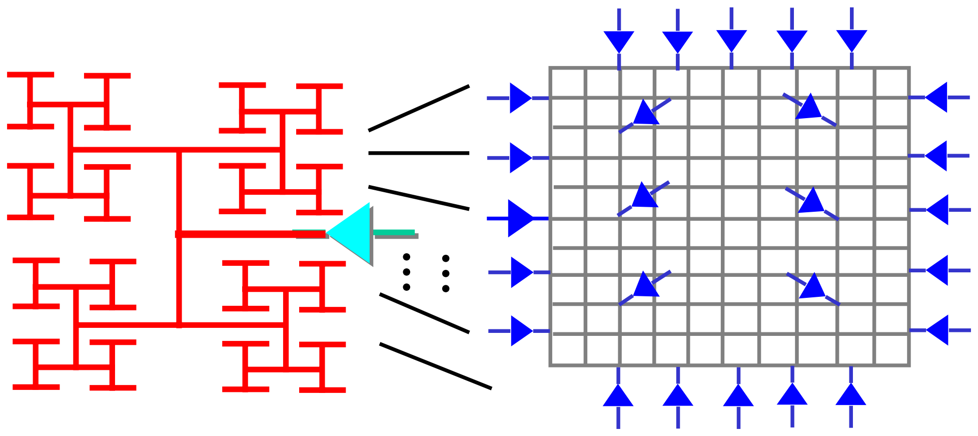
Fig. 52. Non-tree clock distributions.

Although clock meshes are often analyzed in time domain via transient analysis, significant challenges arise due to the large passive mesh structure, which can render the widely used SPICE simulation extremely time consuming, or even impractical. On the other hand, despite that model order reduction techniques have been quite powerful in terms of reducing the complexity of large interconnect analysis problems [93–96], their application is usually limited to networks with a limited number of ports and the extensions to massively coupled mesh structures is nontrivial [97–99].

In contrast, frequency domain steady-state methods, particularly, HB analysis, are specially advantageous in handling passive networks. For example, a large N-port passive mesh network can be directly represented using transfer function matrices evaluated at a set of clock harmonic frequencies. This fact eliminates the difficult task of generating a compact reduced order mesh model as would be the case of time domain analysis. However, the challenge in HB is to be able to efficiently simulate highly nonlinear problems associated with a large number of coupled clock drivers. Despite the ease in handling passive networks, HB analysis is only considered suitable for mildly nonlinear steady-state problems [35, 60, 61, 100]. There exist techniques to improve the robustness of HB analysis via time domain based preconditioners [101, 102]. However, the use of time domain preconditioners looses the important ability of representing passive networks directly in frequency domain using transfer functions.

In this chapter, it is shown that the proposed parallel HB framework based on the parallelizable hierarchical preconditioning can be applied to efficient large clock mesh analysis. To efficiently compute N-port transfer functions for large clock meshes, a SIMO (single-input-multiple-output) based model reduction approach is proposed to compute required transfer functions on a per port basis. Then, the parallel hierarchically preconditioned HB algorithm provides improved efficiency and robustness for strongly nonlinear clock mesh problems. Numerical examples are included to demonstrate the performance of the proposed approach, whereby, significant runtime speedups over the standard transient analysis have been observed.

B. Computation of mesh transfer functions

A passive clock mesh network can be described using the following circuit equations

$$C\frac{d}{dt} + Gx = Bu,\ y = L^T x, \tag{7.1}$$

where $G, C \in \mathbf{R}^{n\times n}$ describe the resistive and energy storage elements in the circuit, $u \in \mathbf{R}^p$ is the input vector, $x \in \mathbf{R}^n$ is the vector of unknown voltages and currents, and $B = [b_1, b_2, \cdots, b_p], L \in \mathbf{R}^{n\times p}$ are the input and output matrices, respectively.

The matrix transfer function of the circuit is $H(s) = L^T(G + sC)^{-1}B$. This implies that one needs to perform an LU factorization of matrix $G + j2\pi kf_0C$ in order to compute the transfer function at a harmonic frequency kf_0. Hence, if HB analysis is conducted based upon a set of M harmonics, then M (large) matrix factorizations are needed to compute the transfer functions.

To reduce the computational cost, model order reduction is employed to produce compact reduced order models for transfer function computation. Since the port number of the mesh can be large, generation of a multi-port reduced order model using a standard algorithm such as PRIMA [96] is challenging. To control the modeling complexity, instead, reduced order models are produced on a per port basic, i.e., multiple SIMO reduced order models are computed, one for each port. A projection-based reduced order model for the i-th input can be generated by computing an orthonormal basis V of the Krylov subspace spanned by $colspan\{r_i, Ar_i, A^2r_i, \cdots\}$, where $A \equiv -G^{-1}C$ and $r_i \equiv G^{-1}B$, and A^kr_i is the k-th order transfer function moment for input i. The SIMO reduced order model is given by a set of projected system matrices

$$\tilde{G} = V^TGV, \tilde{C} = V^TCV, \tilde{b} = V^Tb_i, \tilde{L} = V^TL. \tag{7.2}$$

Once the reduced order model is computed, the transfer functions between any port and port i at all harmonic frequencies can be computed efficiently by performing multiple AC analysis using the small reduced order model. The dominant cost in this SIMO based approach is the LU factorization of the (large) G matrix. However, this is a one-time cost and the same LU factors are reused between all SIMO reduced order models.

C. Clock mesh analysis via harmonic balance

In this section, it is shown how the proposed hierarchical preconditioning based HB algorithm [49,64] can be adopted to efficiently simulate highly nonlinear clock meshes while avoiding the challenges of time domain methods.

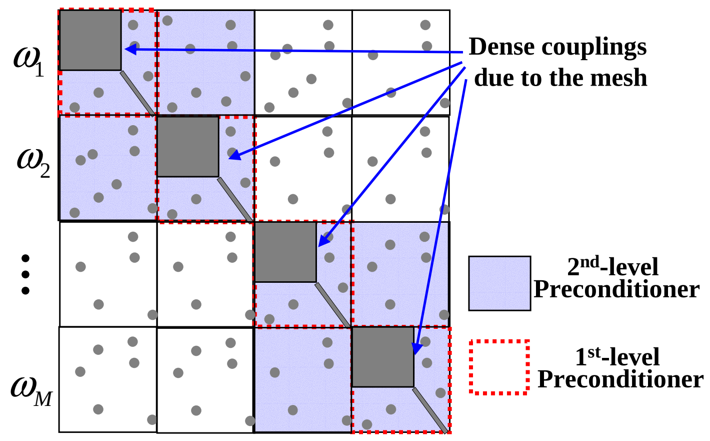

Fig. 53. Non-zero patterns in the Jacobian matrix.

The non-zero patterns of the Jacobian matrix of a typical clock mesh is shown in Fig. 53, where the major index for variable ordering is the frequency. The dense blocks along the diagonal are contributed by the transfer functions of the passive mesh structure. These full blocks have a dimension $N_p \times N_p$, where N_p is the number of nonlinear clock driver ports, which is in the range of a few tens or hundreds. Since these blocks have a limited dimension and only appear along the diagonal, they do not present practical challenges in the iterative solution of the HB problem. Nonlinear devices such as MOS transistors also introduce non-zero patterns into the Jacobian matrix, which are illustrated using "circles". Different from the non-zeros corresponding to the passive network, these non-zeros are sparsely scattered due to the problem sparsity within the nonlinear portion of the circuit. However, these entries are not constrained within the diagonal blocks, they are also present in the off-diagonal blocks manifesting the coupling between different frequency components created by circuit nonlinearities.

For strongly nonlinear circuits, these off-diagonal blocks may contain entries with large magnitude and discarding off-diagonal blocks in the preconditioner can lead to simulation divergence. As explained in the previous chapter, for the proposed parallel HB technique based on hierarchical preconditioning, this difficulty is coped in a multi-level preconditioning way. These multi-level preconditioners are created in the same fashion as that of the top-level problem by recursively decomposing a large block into smaller ones until the block size is small enough for a direct solve as shown in Fig. 9.

D. Experimental results

The proposed parallel HB framework has been applied to validate the performance for clock mesh application. The HB engine is setup for running on 3 PEs in parallel.

First, a mesh with 13k elements including resistors, capacitors and inductors, driven by 17 clock buffers is considered. The time domain response at one sink node of full transient simulation is compared with that of hierarchical HB simulation in Fig. 54 and the region of the dotted rectangle of Fig. 54 is zoomed-in in Fig. 55. From Fig. 54, it can be observed that the proposed hierarchical HB method is fairly accurate.

Next, a larger mesh with 27k elements including resistors, capacitors and inductors, driven by 53 clock buffers is considered. The time domain response at one node of full transient simulation is also compared with that of hierarchical HB simulation in Fig. 56. The zoomed-in region of the dotted rectangle is shown in Fig. 57.

In Table. XVIII, the comparison results between the full transient simulation of five clock cycles and the hierarchical HB simulation based on SIMO reduced order

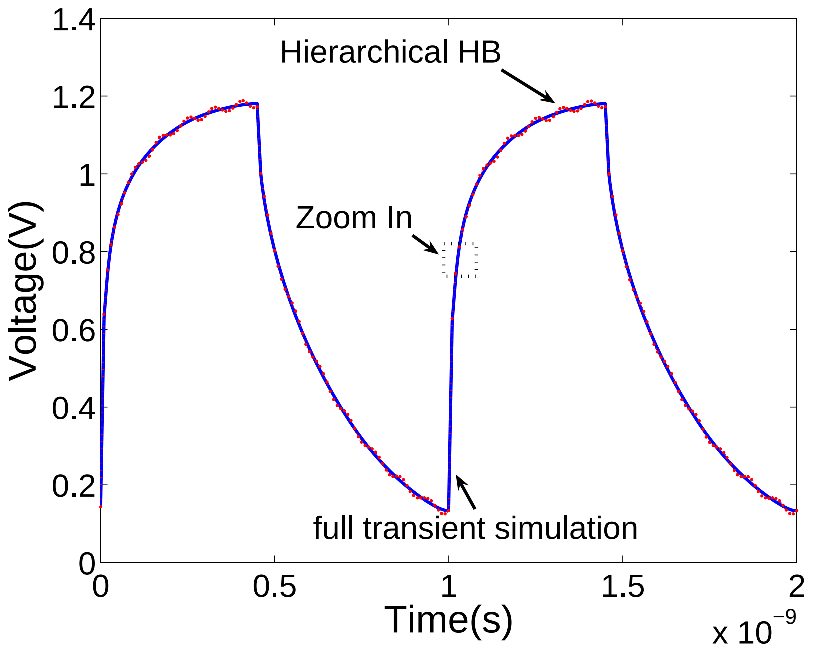

Fig. 54. Comparison between transient simulation & hierarchical HB simulation for mesh1 [full waveform view].

models are listed for different mesh cases. The HB simulation in this table is based on the 3-PE parallel implementation. From the table, a significant speedup for the proposed method can be observed. It can be also observed that the simulation runtimes of mesh3, mesh4 and mesh5 are quite close. This confirms the expectation that the overall complexity of the proposed HB approach is predominately determined by the number of nonlinear clock drivers and has very little dependency on the actual mesh size. A larger mesh size will only contribute to a somewhat higher cost in SIMO-based transfer function computation. This underscores the good scalability of the proposed approach with respect to the increase of mesh complexity. In contrast, the mesh size has a significant impact on the runtime of time-domain transient analysis.

To see the benefits of parallel processing, the serial version of the proposed algorithm is applied to mesh 1. The resulting runtime is 409s, which is 1.7x as much

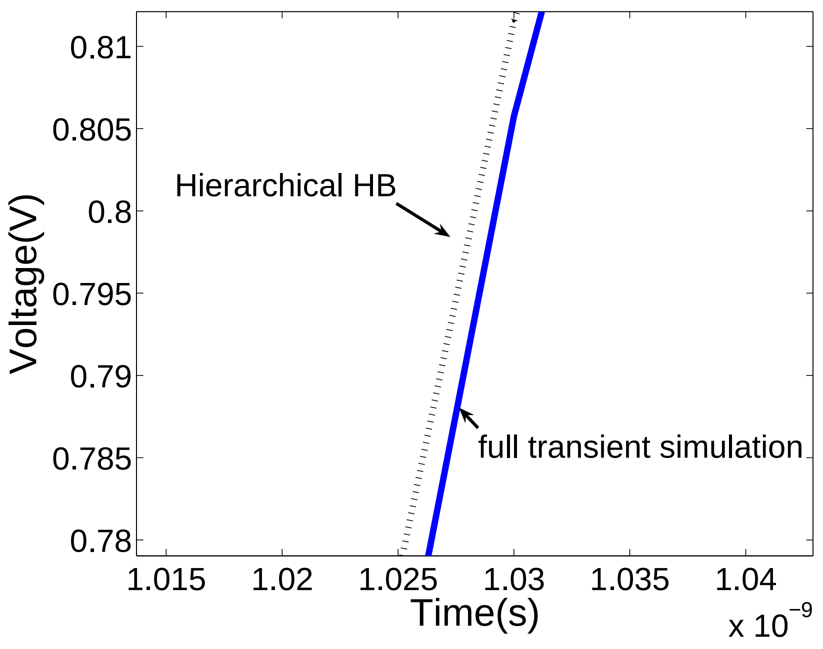

Fig. 55. Comparison between transient simulation & hierarchical HB simulation for mesh1 [zoomed-in view].

as that of the 3-PE parallel processing. For larger mesh designers, it is expected that more pronounced runtime improvement can be achieved by using a larger set of processing elements.

E. Summary

In this chapter, the proposed HB technique based on hierarchical preconditioning has been employed for clock meshes analysis. The efficiency of this HB application stems from the ease in handling large passive mesh structures via transfer functions inherent to HB as well as the improved efficiency brought by the proposed parallel HB simulation technique. The experiments have shown that a significant speedup can be achieved by the proposed HB technique over the full transient simulation.

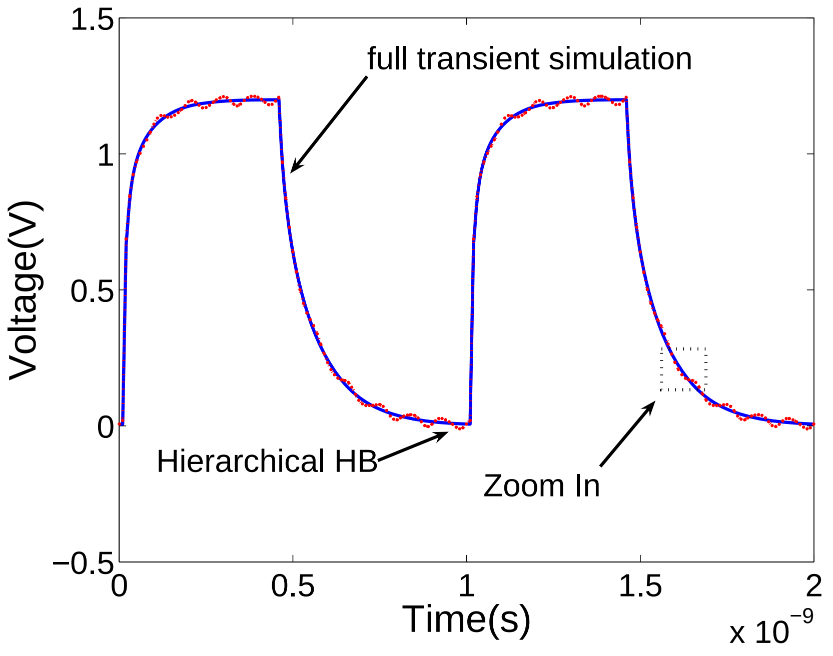

Fig. 56. Comparison between transient simulation & hierarchical HB simulation for mesh2 [full waveform view].

Table XVIII. Comparison for full transient simulation and proposed parallel HB simulation.

Mesh	Mesh	Driver	Full Sim	HB Sim with SIMO		
Case	Size	Num	Sim.T	Gen.T	Sim.T	ave. err
mesh1	13k	17	436s	15.623s	236s	0.29ps
mesh2	27k	53	4h12m	126.26s	21m39s	0.35ps
mesh3	36k	60	4h53m	203.18s	27m45s	0.40ps
mesh4	100k	60	-	400.98s	28m53s	-
mesh5	200k	60	-	905.58s	30m01s	-

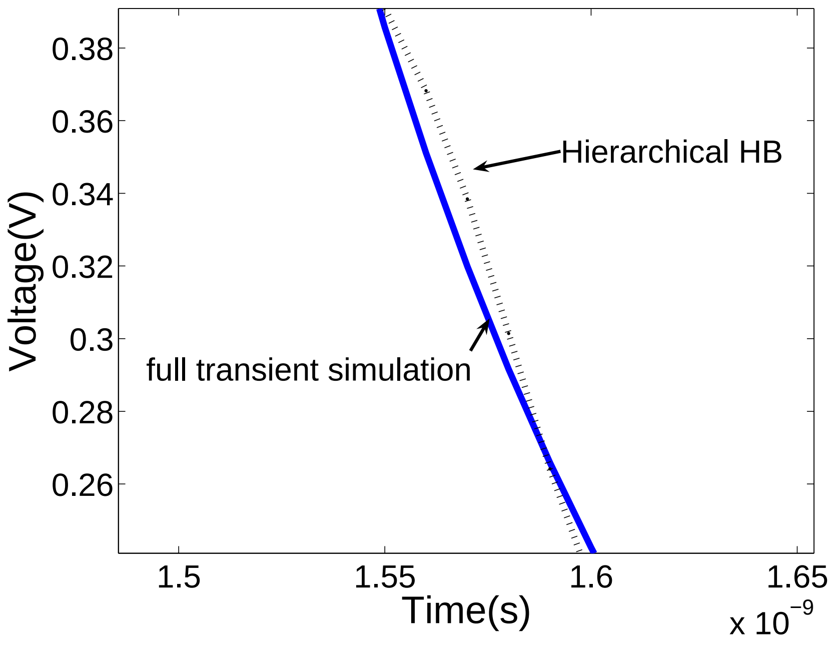

Fig. 57. Comparison between transient simulation & hierarchical HB simulation for mesh2 [zoomed-in view].

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

A. Conclusions

In this dissertation, the parallel algorithms for time and frequency domain circuit simulation are researched. By addressing the existing limitations of well-known circuit simulation techniques and by taking advantage of emerging multicore-based / distributed computing platform, the proposed parallel circuit simulation techniques and their implementations can significantly improve the performance of circuit simulation. The contributions in this research work can be concluded into the following three categories.

Firstly, a new methodology for parallel transient circuit simulation has been proposed for general circuit simulation. The new approach *WavePipe* exploits the hidden high-level parallelism potentials by simultaneously computing circuit solutions at multiple adjacent time points in a way resembling hardware pipelining to boost transient simulations. Different from some existing methods, *WavePipe* facilitates parallel circuit simulations without jeopardying convergence and accuracy. As a coarse-grained parallel approach, *WavePipe* not only requires low parallel programming effort, but also creates new avenues to fully utilize increasingly parallel hardware by going beyond conventional finer grained parallel techniques.

Secondly, from a different angle to speedup transient simulation, an explicit telescopic projective integration based transient simulation technique has been suggested. Due to the improved stability property of telescopic projective integration, the time step of transient simulation is no longer be limited by the smallest time constant, which avoids the stability limitation in many explicit integration methods. From the experimental results, it can be seen that the new approach not only leads to noticeable efficiency improvement in circuit simulation, but also lends itself to straightforward parallelization due to its explicit nature.

Thirdly, a parallel framework for frequency-domain steady-state and envelope-following analyses has been introduced. This framework is constructed based on a naturally-parallelizable preconditioning technique that speeds up the core computation in HB based analysis. Combined with conventional parallel operations, such as parallel device model evaluation, parallel fast fourier transform (FFT) and parallel matrix-vector product, the parallel preconditioning technique can contribute significant runtime speedups for both the steady-state simulation and envelope-following simulation no matter in MPI-based implementation or in multithreading-based implementation from the experimental results.

B. Future work

It has been observed that the proposed parallel algorithms, parallel frameworks and their implementations in this dissertation can bring pretty good performances on current multicore platform (also on MPI-based computing platform) in the experiments. However, the emerging trend on multicore will never stop. With the continuous advance in microprocessor design, it is expected that more and more cores would be integrated in one chip. In the future, the number of cores per chip would be thousands and even more instead of a few or tens. This significant change from multicore to manycore will surely affect the parallel circuit simulations in future. Actually, there exist some prototypes of manycore platform nowadays. For example, Nvidia's Graphic Processing Units (GPUs) have already integrated more than one hundred processing elements on a single die, though processing elements may be not for general purpose. It is important to provide continuous performance improvement with the increase of the number of cores. Therefore, it deserves further research on the improvement of the scalability of the proposed parallel simulation techniques. However, it is difficult for one certain parallel technique to always have a good parallelization scalability.

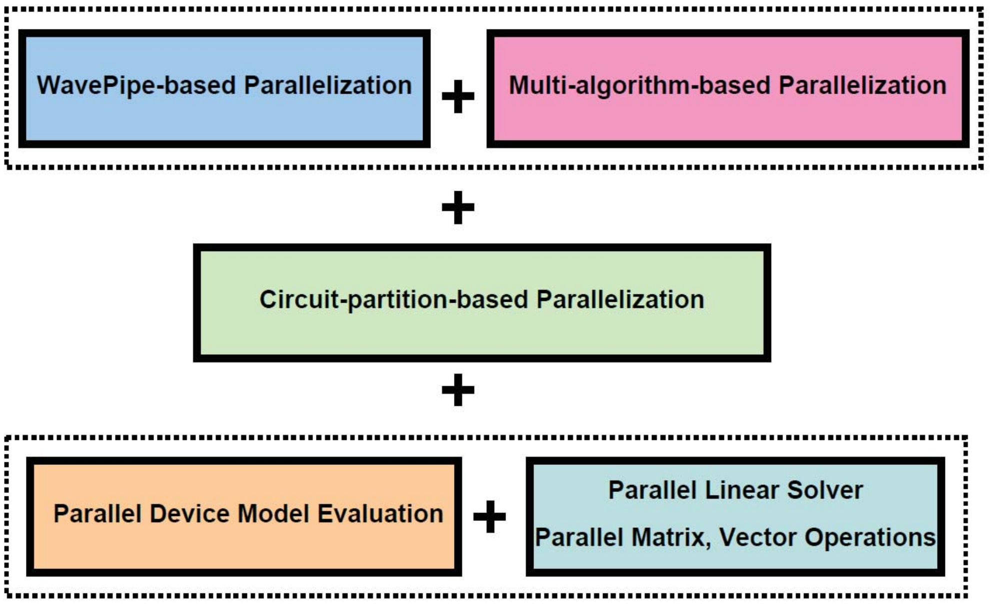
WavePipe-based Parallelization
+
Multi-algorithm-based Parallelization
+
Circuit-partition-based Parallelization
+
Parallel Device Model Evaluation
+
Parallel Linear Solver
Parallel Matrix, Vector Operations

Fig. 58. Combinations of multiple parallelization techniques.

As an alternative, since there already exist many different parallel techniques with different granularities to parallelize circuit simulations, it is interesting to implement the massive parallelization of circuit simulation by combining multiple parallel techniques together as shown in Fig.58, where the coarse-grained WavePipe parallelization [48], the multi-algorithm parallelization [70], the circuit partitioning based technique and the low-level fine-grained parallel techniques such as parallel device mode evaluation and parallel matrix operations are employed together. Correspondingly, it may introduce many new problems, such as how to assign the computing resources to different parallel tasks with different granularity. And it also deserves making efforts to dynamically reallocate processing elements in realtime to balance the workload on computing resources.

For instance, we have parallelized the envelope-following analysis based on parallelized HB technique. Since the envelope-following analysis actually solves a sequence of nonlinear HB problems at each discretized time point along the time axis in a way similar to transient simulation, in order to further realize the massive parallelization of the envelope-following analysis, some coarse-grained parallel techniques and ideas for transient simulation (such as WavePipe method) may be employed and modified for further acceleration.

Another relevant topic is about hybrid programming implementation. With the increase of processing cores, it is expected that the memory hierarchy of the parallel computing platform would be more complex. Because the different parallel programming languages and libraries may specially designed for different parallel computing platforms, in order to make full use of hardware resources, it is meaningful to implement parallel techniques using hybrid programming languages and libraries. For instance, MPI, Pthread and OpenMp may be used together for programming. Therefore, it deserves study on how to map the different parallel tasks and techniques, which may have different parallel granularities, to different parallel models, and furthermore implement the parallel simulation code using such diverse models and libraries.

REFERENCES

[1] J. Friedrich, B. McCredie, N. James, B. Huott, and B. Curran et al, "Design of the Power6^{TM} microprocessor," in *Proc. of IEEE ISSCC*, San Francisco, CA, February 2007, pp. 96–97.

[2] U. Gajanan, M. Hassan, L. Warriner, K. Yen, and B. Upputuri et al, "An 8-core 64-thread 64b power-efficient SPARC SoC," in *Proc. of IEEE ISSCC*, San Francisco, CA, February 2007, pp. 108–109.

[3] S. Vangal, J. Howard, G. Ruhl, S. Dighe, and H. Wilson et al, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm cmos," in *Proc. of IEEE ISSCC*, San Francisco, CA, USA, February 2007, pp. 98–99.

[4] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, and N. Bujanos et al, "An integrated quad-core Opteron^{TM} processor," in *Proc. of IEEE ISSCC*, San Francisco, CA, February 2007, pp. 102–103.

[5] C. Hughes and T. Hughes, *Professional Multicore Programming: Design and Implementation for C++ Developers*, Hoboken, NJ, Wiley Publishing, Inc., 2008.

[6] B. Catanzaro, K. Keutzer, and B. Su, "Parallelizing cad: A timely research agenda for eda," in *Proc. of IEEE/ACM DAC*, Anaheim, CA, 2008, pp. 12–17.

[7] J. Ogrodzki, *Circuit Simulation Methods and Algorithms*, Boca Raton, FL, CRC Press, Inc., 1994.

[8] P. Rodrigues, *Computer-aided Analysis of Nonlinear Microwave Circuits*, Norwood, MA, Artech House, Inc., 1998.

[9] A. Suarez and R. Quere, *Stability Analysis of Nonlinear Microwave Circuits*, Norwood, MA, Artech House, inc., 2003.

[10] C. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuit System*, vol. 22, no. 6, pp. 504–509, 1975.

[11] D. Pederson, "A historical review of circuit simulation," *IEEE Trans. Circuit System*, vol. 31, no. 1, pp. 103–111, 1984.

[12] T. Parker and L. Chua, *Practical Algorithms for Chaotic Systems*, New York, Springer, inc., 1989.

[13] M. Sobhy and A. Jastrzebsky, "Direct integration methods of nonlinear microwave circuits," in *15th European Microwave Conf.*, Europe, 1985, pp. 1110–1118.

[14] I. Maio and F. Canavero, "Differential-difference equations for the transient simulation of lossy mtls," in *ISCAS*, Seattle, WA, 1995, pp. 1402–1415.

[15] M. Biey, F. Bonani, M. Gilli, and I. Maio, "Qualitative analysis of the dynamics of the time-delayed chua's circuit," *IEEE Trans. Circuit System I*, vol. 44, no. 6, pp. 486–500, 1997.

[16] M. Biey, F. Bonani, M. Gilli, and I. Maio, "Influence of the parasitics on the time delayed chua's circuit," in *Electrotechnical Conf.*, Bari, Italy, 1996, pp. 443–446.

[17] K. Kundert and A. Sangiovanni-Vicentelli, "Finding the steady-state response of analog and microwave circuits," in *Proc. of IEEE CICC*, San Jose, CA, 1998, pp. 6.11–6.17.

[18] J. Bonet, P. Pala, and J. Milo, “A discrete-time approach to the steady state analysis of distributed nonlinear autonomous circuits,” in *ISCAS*, Monterey, CA, 1998, pp. 460–464.

[19] L. Nagel, “Spice2: A computer program to simulate semiconductor circuits,” in *Memo UCB/ERL M520*. Electronics Research Lab., Unv. Calif. Berkeley, May 1975, pp. 319–326.

[20] C. Camacho-Penalosa, “Numerical steady-state analysis of nonlinear microwave circuits with periodic excitation,” *IEEE Trans. Microwave Theory Tech.*, vol. 31, no. 9, pp. 724–730, 1983.

[21] V. Rizzoli and A. Neri, “State of the art and present trends in nonlinear microwave cad techniques,” *IEEE Trans. Microwave Theory Tech.*, vol. 36, no. 2, pp. 343–356, 1988.

[22] R. Quere, “Large signal design of broadband monolithic microwave frequency dividers and phase-locked oscillators,” *IEEE Trans. Microwave Theory Tech.*, vol. 41, no. 11, pp. 1928–1938, 1993.

[23] A. Ushida, “Frequency-domain analysis of nonlinear circuits driven by multi-tone signals,” *IEEE Trans. Circuits Syst.-I*, vol. 31, no. 9, pp. 766–778, 1984.

[24] D. Hente and R. Jansen, “Frequency-domain continuation method for the analysis and stability investigation of nonlinear microwave circuits,” *IEE Proceedings*, , no. 6, pp. 351–362, June 1986.

[25] K. Kundert, G. Sorkin, and A. Sangiovanni-Vicentelli, “Applying harmonic balance to almost periodic signals,” *IEEE Trans. Microwave Theory Tech.*, vol. 36, no. 2, pp. 366–378, 1988.

[26] E. Ngoya, J. Rousset, M. Gayral, R. Quere, and J. Obregon et al., "Efficient algorithms for spectra calculations in nonlinear microwave circuit simulators," *IEEE Trans. Circuit Syst.*, vol. 37, no. 11, pp. 1339–1353, 1990.

[27] V. Rizzoli, A. Neri, F. Mastri, and A. Lipparini, "The exploration of sparse-matrix techniques in conjunction with the piecewise harmonic balance method for nonlinear microwave circuit analysis," in *IEEE MTT-S Digest*, Dallas, TX, May 1990, pp. 1295–1298.

[28] H. Brachtendorf, G. Welsch, and R. Laur, "Fast simulation of the steady-state of circuits by the harmonic balance technique," in *ISCAS*, Seattle, WA, May 1995, pp. 1388–1391.

[29] H. Dag and F. Alvarado, "Computation-free preconditioners for the parallel solution of power system problems," *IEEE Trans. Circuits Syst.*, vol. 12, no. 2, pp. 585–591, 1997.

[30] R. Freund, "Passive reduced-order modelling via krylov-subspace methods," in *Proc. IEEE Int. Symp. Computer-Aided Control Syst. Design*, Anchorage, Alaska, 2000, pp. 261–266.

[31] R. Melville and H. Brachtendorf, "An effective procedure for multi-tone steady-state analysis of mixers," in *Proc. of IEEE International Conference on Electronics, Circuits and Sytems*, Malta, Sept 2001, pp. 1449–1453.

[32] K. Mayaram, D. Lee, S. Moinian, D. Rich, and J. Roychowdhury, "Computer-aided circuit analysis tools for rfic simulation: Algorithms, features and limitations," *IEEE Trans. Circuits Syst.*, vol. 47, no. 4, pp. 274–286, 1997.

[33] E. Ngoya and R. Larcheveque, "Envelop transient analysis: A new method for

the transient and steady state analysis of microwave communication circuits and systems," in *Proc. of IEEE MTT-S Int. Microwave Symposium*, San Francisco, CA, June 1996, vol. 3, pp. 1365–1368.

[34] H. Brachtendorf, G. Welsch, and R. Laur, "A novel time-frequency algorithm for the simulation of the steady state of circuits driven by multi-tone signals," in *ISCAS*, Hong Kong, China, 1997, pp. 1508–1511.

[35] J. Roychowdhury, "Efficient methods for simulating highly nonlinear multi-rate circuits," in *Proc. of IEEE/ACM DAC*, Anaheim, CA, June 1997, pp. 269–274.

[36] H. Brachtendorf, G. Welsch, and R. Laur, "A time-frequency algorithm for the simulation of the initial transient response of oscillators," in *ISCAS*, Monterey, CA, 1998, pp. 236–239.

[37] E. Ngoya, J. Rousset, and D. Argollo, "Rigorous rf and microwave oscillator phase noise calculation by the envelope transient technique," in *MTT-S*, Boston, MA, June 2000, pp. 91–94.

[38] E. Lelarasmee, A. Ruehli, and A. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large scale integrated circuits," *TCAD*, vol. 1, no. 3, pp. 131–145, July 1982.

[39] J. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits*, Boston, MA, Kluwer Academic Publishers, 1987.

[40] B. Troyanovsky, Z. Yu, L. So, and R. Dutton, "Relaxation-based harmonic balance technique for semiconductor device simulation," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, Nov 1995, pp. 700–703.

[41] M. Reichelt, A. Lumsdaine, and J. White, "Accelerated waveform methods for parallel transient simulation of semiconductor devices," in *Proc. of IEEE/ACM ICCAD*, Santa Clara, CA, Nov. 1993, pp. 270–274.

[42] U. Wever and Q. Zheng, "Parallel transient analysis for circuit simulation," in *HICSS*, Maui, Hawaii, January 1996, pp. 442–447.

[43] K. Sun, Q. Zhou, K. Mohanram, and D. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, Nov. 2007, pp. 54–59.

[44] D. Rhodes and B. Perlman, "Parallel computation for microwave circuit simulation," *IEEE Trans. on Microwave Theory and Techniques*, vol. 45, no. 5, pp. 587–592, May 1997.

[45] D. Rhodes and A. Gerasoulis, "Scalable parallelization of harmonic balance simulation," in *IPPS/SPDP Workshops*, San Juan, Puerto Rico, 1999, pp. 1055–1064.

[46] D. Rhodes and A. Gerasoulis, "A scheduling approach to parallel harmonic balance simulation," *Concurrency: Practice and Experience*, vol. 12, no. 2-3, pp. 175–187, – 2000.

[47] V. Karanko and M. Honkala, "A parallel harmonic balance simulator for shared memory multicomputers," in *Microwave Conference, 34th European*, Amsterdam, Netherlands, Oct. 2004, pp. 849–851.

[48] W. Dong, P. Li, and X. Ye, "Wavepipe: Parallel transient simulation of analog and digital circuits on multi-core shared-memory machines," in *Proc. of IEEE/ACM DAC*, Anaheim, CA, 2008, pp. 238–243.

[49] W. Dong and P. Li, "Accelerating harmonic balance simulation using efficient parallelizable hierarchical preconditioning," in *Proc. of IEEE/ACM DAC*, San Diego, CA, June 2007, pp. 436–439.

[50] W. Dong and P. Li, "A parallel harmonic balance approach to steady-state and envelope-following simulation of driven and autonomous circuits," *TCAD*, vol. 28, no. 4, April 2009.

[51] W. Dong, P. Li, and X. Ye, "Efficient frequency-domain simulation of massive clock meshes using parallel harmonic balance," in *Proc. of IEEE CICC*, San Jose, CA, Sept. 2007, pp. 631–634.

[52] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing: Design And Analysis of Algorithms*, Essex, England, Addison Wesley, 2004.

[53] W. Gropp and E. Lusk, *User's Guide for mpich, a Portable Implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory, Chicago, IL, 1996, ANL-96/6.

[54] M. Quinn, *Parallel programming in C with MPI and OpenMP*, New York, McGraw-Hill inc., 2004.

[55] G. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *AFIPS Conference Proceedings*, Washington D.C, 1967, pp. 483–485.

[56] J. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[57] K. Alan and F. Horace, "Measuring parallel processor performance," *Communications of the ACM*, vol. 33, no. 5, pp. 539–543, May 1990.

[58] J. Butcher, *Numerical Methods for Ordinary Differential Equations*, West Sussex, England, John Wiley and Sons Ltd., 2008.

[59] K. Kundert, *The Designer's Guide to Spice and Spectre*, Norwell, MA, Kluwer Academic Publishers, 1995.

[60] K. Kundert, J. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*, Boston, MA, Kluwer Academic Publisher, 1990.

[61] P. Feldmann, R. Melville, and D. Long, "Efficient frequency domain analysis of large nonlinear analog circuits," in *Proc. of IEEE CICC*, San Diego, CA, May 1996, pp. 461–464.

[62] Y. Saad, "A flexible inner-outer preconditioned gmres algorithm," *SIAM J. Sci. Comput.*, vol. 14, no. 2, pp. 461–469, 1993.

[63] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Philadelphia, PA, Society for Industrial and Applied Mathematics, 1996.

[64] P. Li and L. Pileggi, "Efficient harmonic balance simulation using multi-level frequency decomposition," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, Nov 2004, pp. 677–682.

[65] W. Dong and P. Li, "Hierarchical harmonic-balance methods for frequency-domain analog-circuit analysis," *TCAD*, vol. 26, no. 12, pp. 2089–2101, Dec 2007.

[66] L. Pillage, R. Rohrer, and C. Visweswariah, *Electronic Circuit and System Simulation Methods,*, New York, McGraw-Hill, 1995.

[67] V. Litovski and M. Zwolinski, *VLSI Circuit Simulation and Optimization*, New York, Chapman & Hall, 1997.

[68] H. Shichman, "Integration system of a nonlinear network analysis program," *IEEE Trans. on Circuit Theory*, vol. CT-17, no. 3, pp. 378–386, August 1970.

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

[70] X. Ye, W. Dong, P. Li, and S. Nassif, "Maps: Multi-algorithm parallel circuit simulation," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, November 2008, pp. 73–78.

[71] A. Devgan and R. Rohrer, "Adaptively controlled explicit simulation," *IEEE Trans. on CAD*, vol. 13, no. 6, pp. 746–762, June 1994.

[72] R. Griffith and M. Nakhla, "A new high-order absolutely-stable explicit numerical integration algorithm for the time-domain simulation of nonlinear circuits," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, 1997, pp. 276–280.

[73] C. Gear and I. Kevrekidis, "Projective methods for stiff differential equations: Problems with gaps in their eigenvalue spectrum," *SIAM J. Sci. Comput.*, vol. 24, no. 4, pp. 1091–1106, 2002.

[74] C. Gear and I. Kevrekidis, "Telescopic projective methods for parabolic differential equations," *J. Comput. Phys.*, vol. 187, no. 1, pp. 95–109, 2003.

[75] K. Mayaram, P. Yang, R. Burch J. Chern, L. Arledge, and P. Cox, "A parallel block-diagonal preconditioned conjugate-gradient solution algorithm for circuit and device simulations," in *Proc. of IEEE/ACM ICCAD*, Santa Clara, CA, Nov. 1990, pp. 446–449.

[76] M. Sosonkina, D. Allison, and L. Watson, "Scalable parallel implementations of the gmres algorithm via householder reflections," in *International Conference on Parallel Processing*, Minneapolis, MN, 1998, p. 396.

[77] A. Basermann, U. Jaekel, M. Nordhausen, and K. Hachiya, "Parallel iterative solvers for sparse linear systems in circuit simulation," *Future Gener. Comput. Syst.*, vol. 21, no. 8, pp. 1275–1284, 2005.

[78] J. Vetter and B. de Supinski, "Dynamic software testing of MPI applications with Umpire," in *Proceedings of ACM/IEEE conference on Supercomputing, address = "Dallas, TX", year = "2000", url = "citeseer.ist.psu.edu/vetter00dynamic.html"*.

[79] E. Ngoya, A. Suarez, R. Sommet, and R. Quere, "Steady state analysis of free or forced oscillators by harmonic balance and stability investigation of periodic and quasi-periodic regimes," *Int. J. Microw. Millim.-Wave Comput.-Aided Eng.*, vol. 5, no. 3, pp. 210–233, Mar 1995.

[80] M. Gourary, S. Ulyanov, M. Zharov, S. Rusakov, and K.K.Gullapalli et al, "Simulation of high-q oscillators," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, Nov 1998, pp. 162–169.

[81] K. Boianapally, T. Mei, and J. Roychowdhury, "A multi-harmonic probe technique for computing oscillator steady states," in *Proc. of IEEE/ACM ICCAD*, Nov 2005, pp. 610–613.

[82] X. Duan and K. Mayaram, "An efficient and robust method for ring-oscillator simulation using the harmonic-balance method," *TCAD*, vol. 24, no. 8, pp. 1225–1233, 2005.

[83] K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "An envelope-following method for the efficient transient simulation of switching power and filter circuits," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, October 1988, pp. 446–449.

[84] J. White and S. Leeb, "An envelope-following approach to switching power converter simulation," *IEEE Trans. on Power Electronics*, vol. 6, pp. 303–307, April 1991.

[85] L. Silveira, J. White, and S. Leeb, "A modified envelope-following approach to clocked analog circuit simulation," in *Proc. of IEEE/ACM ICCAD*, Santa Clara, CA, 1991, pp. 20–23.

[86] P. Feldmann and J. Roychowdhury, "Computation of circuit waveform envelopes using an efficient, matrix-decomposed harmonic balance algorithm," in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, Nov 1996, pp. 295–300.

[87] V. Rizzoli, A. Neri, F. Mastri, and A. Lipparini, "A krylov-subspace technique for the simulation of rf/microwave subsystems driven by digitally modulated carriers," *Int. J. RF Microwave Comput.-Aided Eng.*, vol. 9, pp. 490–505, 1999.

[88] V. Rizzoli, A. Costanzo, and F. Mastri, "Efficient krylov-subspace simulation of autonomous rf/microwave circuits driven by digitally modulated carriers," *IEEE Microwave Wireless Comp. Lett.*, vol. 11, no. 7, pp. 308–310, 2001.

[89] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[90] S. Balay, K. Buschelman, W. Gropp, D. Kaushik, and M. Knepley et al, "PETSc Web page," 2001, http://www.mcs.anl.gov/petsc.

[91] P. J. Restle et al., "A clock distribution network for microprocessors," *IEEE J. of Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, May 2001.

[92] P. Restle, T. McNamara, D. Webber, P. Camporese, and K. Eng et al, "The clock distribution of the power4 microprocessor," in *Proc. of IEEE ISSCC*, San Francisco, CA, Feb. 2002, pp. 144–145.

[93] L. Pillage and R. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 352–366, April 1990.

[94] P. Feldmann and R. Freund, "Efficient linear circuit analysis by padé approximation via the lanczos process," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 639–649, May 1995.

[95] L. Silveira, M. Kamon, and J. White, "Efficient reduced-order modeling of frequency-dependent coupling inductances associated with 3-d interconnect structures," in *Proc. of IEEE/ACM DAC*, San Francisco, CA, June 1995, pp. 376–380.

[96] A. Odabasioglu, M. Celik, and L. Pileggi, "Prima: Passive reduced-order interconnect macromodeling algorithm," *IEEE Trans. Computer-Aided Design*, vol. 17, no. 8, pp. 645–654, August 1998.

[97] L. Silveira and J. Phillips, "Exploiting input information in a model reduction algorithm for massively coupled parasitic networks," in *Proc. of IEEE/ACM*

DAC, San Diego, CA, June 2004, pp. 385–388.

[98] P. Feldmann and F. Liu, “Sparse and efficient reduced order modeling of linear subcircuits with large number of terminals,” in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, November 2004, pp. 88–92.

[99] P. Li and W. Shi, “Model order reduction of linear networks with massive ports via frequency-dependent port packing,” in *Proc. of IEEE/ACM DAC*, San Francisco, CA, July 2006, pp. 267–272.

[100] M. Gourary, S. Rusakov, S. Ulyanov, M. Zharov, and K. Gullapalli et al, “The enhancing of efficiency of the harmonic balance analysis by adaptation of preconditioner to circuit nonlinearity,” in *Proc. of IEEE ASP-DAC*, Pacifico Yokohama, Japan, January 2000, pp. 537–540.

[101] O. Nastov, *Spectral Methods for Circuit Analysis*, Ph.D. Dissertation, Dept. of Electrical and Computer Science, MIT, 1999.

[102] F. Veerse, “Efficient iterative time preconditioners for harmonic balance rf circuit simulation,” in *Proc. of IEEE/ACM ICCAD*, San Jose, CA, November 2003, pp. 251–254.

VITA

Wei Dong received the B.E. degree in electrical engineering from Xi'an JiaoTong University, China, the M.E. degree in electrical engineering from Shanghai JiaoTong University, China, and the Ph.D. degree in computer engineering from Texas A&M University. From 2003 to 2005, he was a research faculty member with Institute of Image Communication and Information Processing & the IC and System Research Center in Shanghai JiaoTong University. He also worked as an intern at Texas Instruments in Dallas in the summer of 2008.

His current research interests include VLSI circuit design and computer-aided design with an emphasis on high-performance parallel circuit simulation techniques. He received the Best Paper Award from IEEE/ACM Design Automation Conference (DAC) in 2008 and the Best Paper Award Nomination from IEEE/ACM International Conference on Computer-Aided Design (ICCAD) in 2008, respectively. His mailing address is Department of Electrical and Computer Engineering, Mail Stop 3128, Texas A&M University, College Station, TX, 77843-3128.

The typist for this dissertation was Wei Dong.