Parallel Algorithms for Simulation and Optimization of On-Chip Power Grid Networks

by

Harjot Singh Dhindsa

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Applied Sciences

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Electronics
Faculty of Engineering
Carleton University
Ottawa, Ontario, Canada

September 2009

Your file Votre référence
ISBN: 978-0-494-60257-7
Our file Notre référence
ISBN: 978-0-494-60257-7

The undersigned hereby recommends

To The Faculty of Graduate Studies and Research Office

acceptance of the thesis

Parallel Algorithms for Simulation and Optimization of On-Chip Power Grid Networks

Submitted by

Harjot Singh Dhindsa, B.Eng.

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical Engineering

Professor Q.J. Zhang
Chair, Department of Electronics

Professor Ram Achar
Thesis Co-Supervisor

Professor Michel Nakhla
Thesis Co-Supervisor

Carleton University
September 2009

Abstract

In modern integrated circuits, aggressive device scaling, higher operating frequencies and increased circuit complexity have caused an increase in current and power needs while decrease in voltage levels. Consequently, power integrity design and analysis presents a major challenge in modern semiconductor designs and systems.

The primary problem with the analysis of on-chip power distribution networks (PDNs) is their large size. Most conventional circuit simulation techniques prove to be inadequate for analysis of a network of this size. This warrants development of new simulation methodology adapted to handling large problem size. Furthermore, in recent years, parallel platforms are emerging as a powerful computation medium promising unprecedented processing ability in least possible time. Also, recently waveform relaxation (WR) techniques have been proposed for efficient analysis of power distribution networks on serial platforms. The waveform relaxation techniques can be very easily parallelized to provide fast scalable parallel algorithms for power grid analysis.

Acknowledgments

I would like to thank my supervisors, Professors Ram Achar and Michel Nakhla, who provided the initial inspiration for this work and gave critical guidance throughout the process. Their solid understanding of the subject and mentorship helped me grow and mature as a serious graduate researcher and made this thesis an enjoyable experience.

I thank Koko Mihan of Javelin Systems, Dr. Eli Chiprout of Intel and Dr. Albert Ruehli of IBM for their valuable comments and ideas, which were indispensable in producing this thesis.

I would like to acknowledge and thank my colleagues, D. Saraswat, B. Nouri, A. Charest, N. Nakhla, A. Narayanan, M. Adel, A. Jerome, V. Ambalavanar, M. Siddique, Y. Hong, D. Trifkovic, N.Sovieko and L. Filipovic for the friendly, often philosophical, discussions which made me laugh and kept me engaged. I would also like to express my gratitude to the professors, staff and students at the department of electronics for making my work an enjoyable experience.

I leave a special note of thanks to my friend and colleague Arvind Sridhar. Without his assistance and support this thesis would have been incomplete.

I leave a special note of thanks to my friend Amandeep Bhullar, and his parents. Without their affection and encouragement, working on this thesis would have been very hard.

I am grateful to my friend and colleague Douglas Paul who was patient enough to answer all my questions and helped me navigate smoothly through the course of my research.

I am indebted to my manager at Synopsys Kevin Beaudette. Without his support this thesis would have been incomplete.

Finally, I thank my parents Labh and Rajinder Dhindsa, and my brother, Tegbir, for their years of patience. They inculcated in me, the determination and dedication to do whatever I undertake well. I am grateful to you for loving me and supporting my throughout this thesis.

Dedicated to my parents

Labh and Rajinder Dhindsa

for their affection and encouragement

Contents

List of Figures

List of Tables

List of Symbols

α	Order of computational complexity of a sparse matrix technique, while solving a system of linear equations
c_i	Capacitance connected at node i
g_{ij}	Value of the conductance between node i and node j
h	Step-size used in the backward Euler method
s	Frequency in the laplace domain
ω	Frequency in radians
t	Time
$i(t), v(t)$	Time-domain current and voltage waveforms, respectively
$\boldsymbol{A}$	MNA matrix
$\boldsymbol{b}$	Selection matrix for the input sources in a MNA system

$\boldsymbol{C}$	Frequency-dependent part of the MNA matrix
$\boldsymbol{F}$	System of equations representing the time-domain behavior of a general dynamical system
f_{max}	Maximum frequency of operation of a microprocessor
$\boldsymbol{G}$	Frequency-independent part of the MNA matrix
Gnd	Ground terminal in an integrated circuit
I_{WR}	Waveform relaxation current source
I_{WRS}	Waveform relaxation current source in sensitivity circuit
N	Number of nodes in a power grid circuit
N_p	Number of number of processors in the given system
N_{BRP}	Number of BRP partitions in a power grid circuit
$N_{S,BRP}$	Number of sensitivity BRP partitions in a power grid circuit
N_O	Number of odd BRP partitions in a power grid circuit
$N_{S,O}$	Number of odd sensitivity BRP partitions in a power grid circuit
N_E	Number of even BRP partitions in a power grid circuit
$N_{S,E}$	Number of even sensitivity BRP partitions in a power grid circuit

$\boldsymbol{u}(t)$	Input vector in a MNA system
$\boldsymbol{X}(t)$	Vector of unknown variables in a MNA system
$\boldsymbol{y}(t)$	Vector of unknown variables in a general dynamical system

Abbreviations

AMG	Algebraic multi-grid
BE	Backward Euler
BGA	Ball grid array
BRP	Block row partitioning scheme – a partitioning technique for simulating power grid circuits proposed in this thesis
C4	Controlled collapse chip connection – a method to make wire connections to an integrated circuit from a flip chip package
CAD	Computer-aided design
ccNUMA	Cache coherent non uniform memory access
CGS	Conventional Gauss-Seidel method – an iterative technique to solve for a system of linear equations
CMOS	Complementary metal oxide semiconductor – a class of integrated circuits

CPU	Central processing unit
DMA	Distributed memory architecture
FB	Forward backward - substitution method used to solve matrices
GJ	Gauss-Jacobi method – an iterative technique to solve for a system of linear equations
GS	Gauss-Seidel method – an iterative technique to solve for a system of linear equations
HPCVL	High performance computing virtual laboratory
IC	Integrated circuit
IFFT	Inverse fast fourier transform
KCL	Kirchhoff's current law
LU	Lower upper - matrix decomposition
MNA	Modified nodal analysis
MPI	Message passing interface
MTL	Multiconductor transmission lines
NMOS	n-channel metal oxide semiconductor transistor

NUMA Non uniform memory access

OpenMP Open multi-processing

PCB Printed circuit board

PDN Power distribution network

PGA Pin grid array

PMOS p-channel metal oxide semiconductor transistor

RAM Random access memory

SMA Shared memory architecture

SMP Symmetric multi-processing

SOR Successive over-relaxation method – an iterative technique to solve for a system of linear equations

SRP Single row partitioning scheme – a partitioning technique for simulating power grid circuits proposed in this thesis

TP Transverse partitioning scheme – a partitioning technique for simulating multiconductor transmission lines

UMA Uniform memory access

VLSI	Very large scale integration
VRM	Voltage regulator module
WR	Waveform relaxation

Chapter 1

Introduction

1.1 Background and Motivation

Technology scaling has been the principal driver behind improving the performance characteristics of integrated circuits (ICs) over the past several decades. There has also been significant improvement in speed and integration density of the ICs. These factors have made managing the on-chip power distribution a difficult task. Modern ICs operate at high frequencies which has increased the distributed current to tens of amperes, while the noise margin of the power supply has been shrunk consistent with decreasing power supply levels. These trends have elevated the problem of power distribution to the forefront of challenges in developing high performance ICs.

The biggest challenge in the analysis of power distribution networks is the size

of the problem which is in the range of few millions of nodes. The sheer size of the problem renders most of the conventional circuit analysis techniques ineffective. Recently, waveform relaxation techniques have been successfully applied to analysis of on-chip power distribution networks providing significant computational advantages over conventional techniques. In spite of the recent advances, designers are seeking further reduction in simulation times to enable interactive design process.

Also, in the recent years, high-performance parallel computing is becoming an attractive option for scientists and engineers to solve large and complex problems. Parallel processing is progressively being seen as a lucrative method for the fast solution of computationally large and data-intensive applications. The emergence of inexpensive parallel computers such as desktop multiprocessors and PC clusters has set a stage for significant need to develop parallel algorithms for CAD applications.

In this thesis new algorithms are developed for fast parallel simulation of power grid networks. The specific contributions are listed below.

1.2 Contributions

The major subject of this thesis is the development of efficient scalable parallel algorithms for on-chip power distribution analysis. The proposed algorithms provide fast, accurate parallel solutions for on-chip power grid analysis which can be incorporated in SPICE like simulators. The main contributions of the thesis are as follows.

1. **Parallel power grid transient analysis**: Novel parallel algorithms employing waveform relaxation method are developed for transient analysis of large on-chip power distribution networks. For this purpose, parallel Gauss-Seidel waveform relaxation and performance enhancing pipelining techniques have been developed. The parallel algorithms provide scalable performance in shared memory environment [1–3].

2. **Parallel sensitivity analysis of power grid networks**: Waveform relaxation technique has been extended for sensitivity analysis of power grid networks. Novel partitioning schemes, similar to the transient analysis, have been employed for sensitivity analysis. Parallel Gauss-Seidel waveform relaxation method and efficient parallel pipelining techniques have also been developed for sensitivity analysis. The developed algorithm exhibits simulation speed-ups with increasing number of processors on multi-core platforms.

1.3 Organization of the Thesis

The thesis is organized as follows. Chapter 2 introduces parallel algorithms design, analysis and system architecture considerations. Chapter 3 provides a background and literature survey of power integrity analysis. Various existing techniques addressing these problems are also examined. Chapter 4 reviews the theory behind waveform

relaxation techniques and provides literature survey for the same. Chapter 5 proposes a parallel waveform relaxation based algorithm for efficient transient analysis of large power grid networks and demonstrates the effectiveness of the algorithm using some numerical examples. Chapter 6 proposes a waveform relaxation based algorithm for parallel sensitivity analysis of on-chip power grid networks. Finally, Chapter 7 gives a brief summary of the thesis and provides directions for future related research.

Chapter 2

Introduction to Parallel Algorithms and Computing

With the rapid advances in VLSI technology, the computational complexity and the problem size to be handled have also increased significantly. Even though the computing machines today are extremely fast, some scientists and engineers still must wait hours or days for their programs and simulations to complete. This thirst for higher computing power has further led to the emergence of parallel computing platforms.

All parallel machines exhibit concurrency, usually performing a lot of operations at the same time. This has considerably reduced the computation time. The scalable performance and lower cost of parallel platforms is demonstrated in wide variety of applications like:

- Electrical Engineering, Circuit Design, Microelectronics
- Databases, data mining
- Medical imaging and diagnosis
- Bioinformatics
- Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
- Advanced graphics and virtual reality
- Chemistry, Molecular Sciences etc.

Due to the recent advances in parallel hardware, the parallel algorithm development has emerged as a key research area for applications which require high computational power. In this chapter, the basics of parallel algorithm development and various performance metrics are discussed. The effects of system architecture and memory are also presented in the end of this chapter.

2.1 Parallel Algorithms: Design and Analysis

Algorithm development and design are key components for extracting maximum performance on a particular platform, serial or parallel. A serial algorithm defines a sequence of steps to solve a problem on a sequential machine. On the other hand, a parallel algorithm defines a sequence of steps to solve a problem on a parallel machine.

But the design of a parallel algorithm has additional constraints compared to serial algorithm like the concurrency and synchronicity specifications at various steps. In general, a parallel algorithm development and design includes [4]:

- Partitioning the problem into smaller tasks,
- Mapping the tasks onto different processors which execute them concurrently,
- Synchronizing the processors at various stages of parallel algorithm execution.

The first and the most important step for a parallel algorithm design is partitioning of the given problem into various tasks. All the tasks so created should be independent of each other so that they can be executed concurrently to reduce the time required to solve the entire problem. However, in general, some tasks may need data from other tasks and thus may require to wait for those tasks to finish execution. Hence, appropriate partitioning strategy should be adopted such that the created tasks should be least dependent on each other.

In parallel algorithms, granularity is the extent to which the problem is decomposed into smaller tasks. A decomposition into a large number of small tasks is called fine-grained parallelism whereas a decomposition into a small number of large tasks is referred to as coarse-grained parallelism. In the context of parallel algorithms, granularity is a qualitative evaluation of the ratio of computation to communication. So, in

a coarse grained parallel algorithm a large amount of computations are done between communication events. On the other hand, a fine grained algorithm requires comparatively less amount of computations to be performed in between communication. It is to be noted that, the type of granularity for parallelizing a problem is determined by the type of the algorithm and the hardware system available. However, for most engineering applications, coarse-grained parallelism is appropriate as it allows each processor to solve a large portion of the problem concurrently. Fine-grained parallelism can be thought of partitioning the program (for example parallelizing each line of code within a program) and course-grained parallelism partitions the overall problem itself.

Mapping tasks onto different processors determines the efficiency of the utilization of concurrency determined by partitioning of the problem. An effective mapping technique should reduce the processor idle time and also reduce the amount of time processors spend communicating with each other. However, both these objectives are in conflict with each other. For instance, the best mapping scheme to reduce the inter processor communication is to map all the tasks onto the same processor. This means that all other processors are idle, which is not desirable. Due to the conflicting nature between these objectives, efficient mapping technique is a nontrivial problem. In [4], various techniques such as static and dynamic mapping are mentioned for optimal mapping of the tasks onto different processors.

The idea of mapping techniques is to balance the computations (work-load) among the processors so that processor idle times are reduced. In static mapping, the tasks are distributed among the processors prior to the execution of the algorithm which makes it easier to design. On the other hand, in dynamic mapping tasks are distributed among the processors during the execution of the algorithm which makes it relatively tougher to design. However, dynamic mapping is more adaptive and hence effective compared to static mapping.

Processor synchronization and data sharing between multiple processors is also crucial for a good parallel algorithm design. Synchronization is the coordination of processors in real time. It is usually accomplished by creating a synchronization point within the problem where a processor may not proceed further until all the other processors reach the same synchronization point.

In the next section, various performance metrics for parallel algorithms have been discussed.

2.2 Performance Metrics for Parallel Algorithms

It is vital to analyze the performance of parallel algorithms to determine the best algorithm, appropriate hardware platform and exploring the full potential of parallelism. For this purpose, various metrics to evaluate the performance of parallel algorithms have been discussed in this section.

Execution Time

The serial run time, for an algorithm, is the time it takes to execute the algorithm from the beginning to end on a sequential platform (or on a single processor). On the other hand, the parallel run time is the time elapsed from the instant parallel computation starts to the instant the last processing element finishes the execution. The serial run time is denoted by T_s and the corresponding parallel run time is denoted by T_p, where p is the number of available processors.

Communication Overhead

In all the parallel algorithms, there is some communication overhead involved. Communication overhead for a parallel algorithm is defined as the total time spent by all the processing elements over and above the time taken by the best known sequential algorithm for the same problem on a sequential platform. In other words, it is the amount of time required to coordinate parallel tasks, as opposed to doing useful work. Communication overhead includes:

- Task start-up time
- Synchronizations
- Data communications
- Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.

- Task termination time

For a p processor system with parallel run time equal to T_p and the corresponding serial run time equal to T_s, the communication overhead (T_o) is given by [4]

$$T_o = pT_p - T_s \tag{2.1}$$

CPU Time vs Wall Time

When measuring parallel performance, it is vital to clarify the difference between cpu time and wall time. CPU time is the definite time spent by the processors doing computations, i.e. it is a measure of work. For example, if 2 processors each spend 3 hours working on a task, then we would say the CPU time is 6 hours (number of CPUs $\times$ work per CPU). CPU time is analogous to the term man hours. For example, if it took 6 months to build a bridge, and there were 30 workers, then we would say it took 180 man hours to complete the bridge. On the other hand, wall time, is the actual elapsed time determined by a watch or a wall clock. Wall time is the time which is relevant when dealing with the speed up of the parallel algorithms.

Speed up

The speed up achieved by a parallel algorithm is one of the simplest and most widely used indicators for parallel performance. It quantifies the relative advantage and

corresponding performance gain achieved by parallelizing a given algorithm over a sequential algorithm. Speed up is measured as the ratio of the time taken for an algorithm to execute on a sequential platform to the time taken for the same algorithm on a parallel platform.

For a p processor system with parallel run time equal to T_p and the corresponding serial run time equal to T_s, the speed up S_p is given by [4]

$$S_p = \frac{T_s}{T_p} \tag{2.2}$$

For the above derivation, it is assumed that the p processors used by the parallel and sequential algorithms are of the same type.

Efficiency

Efficiency is a measure of the utilization efficiency of all the processing elements when the algorithm is executed on a parallel platform. It is defined as [4]

$$E = \frac{S_p}{p} \tag{2.3}$$

where S_p is the speed up and p is the number of processors.

For an ideal parallel algorithm design, we have $S_p = p$ and E=1 (100% efficiency. This means that for a quad-core (4 processor) system, an ideal parallel algorithm design will achieve 4 times the speed up as compared to the serial counterpart such that efficiency is 100%. However, in practice $S_p \leq p$ and $0 \leq E \leq 1$.

Scalability

The ability of a parallel algorithm to provide significant speed up with increasing number of processors is referred to as the scalability of the parallel algorithm. For an ideal case, we expect the algorithm to show an increasing speed up as the number of processors are increased. But in practice, these observations may not be valid. As the number of processors increase, the corresponding inter processor communication and processor idle times also tend to increase. Therefore, if the number of processors are increased beyond a certain number, the overall efficiency of most parallel algorithms goes down. Hence, scalability indicates the ability of a parallel system to effectively utilize the increasing processing resources, compared to the effort wasted in communication and synchronization.

2.3 Speed Up Estimation

Amdahl's Law

Amdahl's Law, named after computer architect Gene Amdahl who formulated it, is a way to calculate the theoretical maximum speed up that can be achieved for an algorithm by using multiple processors. Let f be the fraction of the algorithm that is sequential and cannot be be parallelized. Then the fraction of the algorithm that can be parallelized is $1 - f$. Amdahl's Law states that, for such an algorithm, the

maximum achievable speed up (S_A) is given by [5]

$$S_A = \frac{1}{f + \frac{1-f}{p}}. \tag{2.4}$$

where p is the number of processors.

For equation (2.4), it can be seen that if $p \to \infty$ then $S_A \to \frac{1}{f}$. This implies that the maximum attainable speed is limited by the sequential (unparallelizable) part of the algorithm. Hence, smaller the sequential fraction, higher the speed up that can be achieved through parallelization. For example, if $f = 0.5$ and $p = 4$ the corresponding $S_A = 1.6$. However, for same number of processors if $f = 0.25$ the subsequent $S_A = 2.3$. And if $f = 0$ the subsequent $S_A = 4$ which means ideal speed up. Fig. 2.1 shows the speedup variation for different serial fractions for the given algorithm. Fig. 2.2 shows the speedup for different serial fractions with an increasing number of processors. Clearly, the maximum speed up that can be achieved corresponds to the case where $f = 0$. As seen from the Fig. 2.2, for $f \geq 0.01$, the speedup saturates after a few processors and does not change significantly with any further reduction in f as number of processors increase. This implies that Amdahls law is more suited for systems which are not highly scalable, for instance shared memory systems as it assumes constant problem size. The major limitation to Amdahl's law is that, it assumes that the problem size is constant and does not vary with the number of processors. To address these shortcomings, Gustafsons law was proposed.

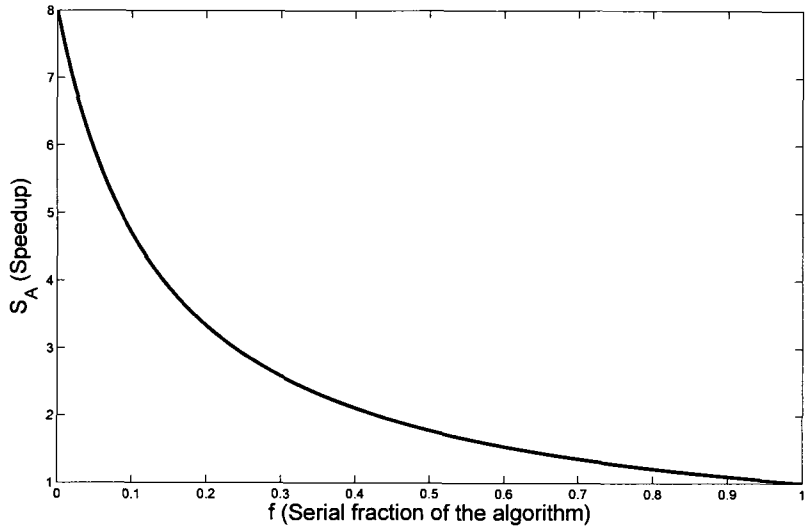

Figure 2.1: Speed Up vs Fraction of the Serial Part Using Amdahl's Law for $p = 8$

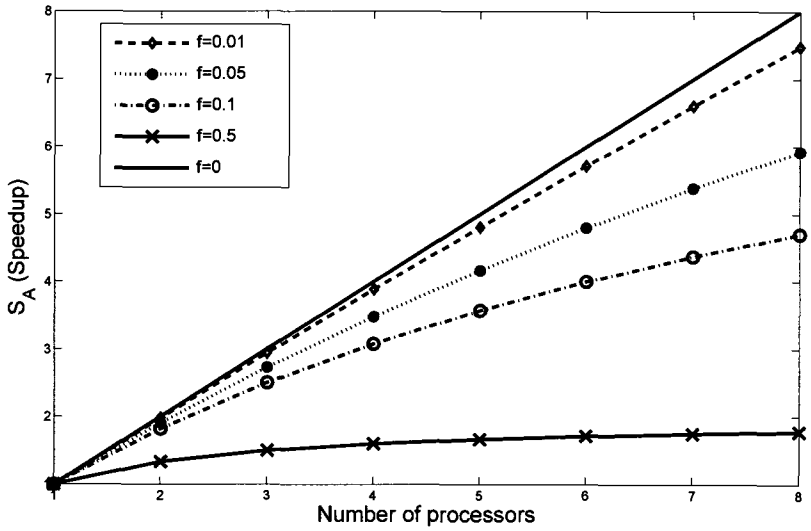

Figure 2.2: Speed Up vs Number of Processors Using Amdahl's Law

Gustafsons Law

In practice, as the number of processors increase the problem size to handle also tends to increase. In other words, the problem size scales with the number of processors. John L. Gustafson from Sandia National Laboratories formulated Gustafsons Law to take this into account [6]. According to this law, the run time is constant (instead of treating the problem size to be constant) such that a scaled speed up is obtained. Gustafson's law is given by [6]

$$S_G = \frac{t_s + (t_p \times p)}{t_s + t_p}, \tag{2.5}$$

where t_s is the time spent to execute serial part of the algorithm on a parallel system, t_p is the time spent to execute the parallel part on the parallel system and p is the number of processors. Assuming $t_s + t_p = 1$, we get [6]

$$\begin{aligned} S_G &= t_s + (t_p \times p) \\ &= p + (1 - p)t_s. \end{aligned} \tag{2.6}$$

From equation (2.6), we see that the parallel time is scaled by p hence the speed up so obtained is called scaled speed up. Fig. 2.3 shows the scaled speed up vs. the serial fraction of the algorithm using Gustafson's law with $p = 8$. As seen from the figure, the speed up is a straight line with slope $1 - p$. Notice that the speed up numbers are better than what predicted by Amdahl's law. Moreover, the speed up grows linearly with the increasing number of processors as shown in Fig. 2.4. The

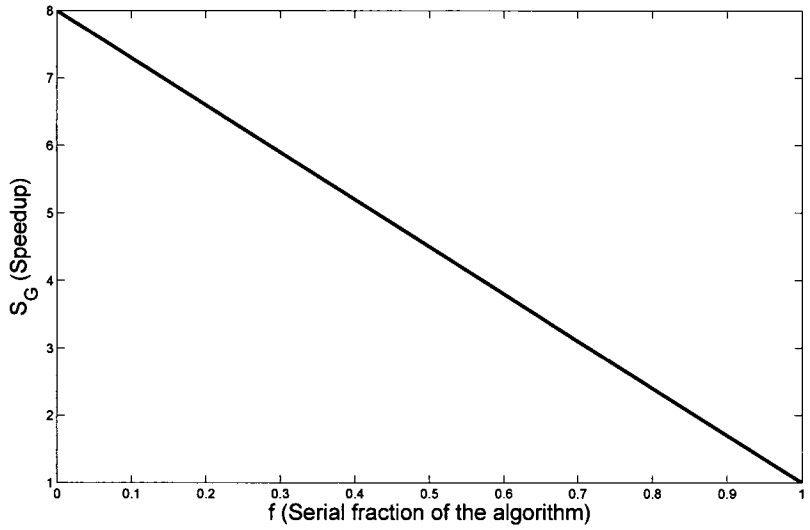

Figure 2.3: Speed Up vs Fraction of the Serial Part Using Gustafson's Law

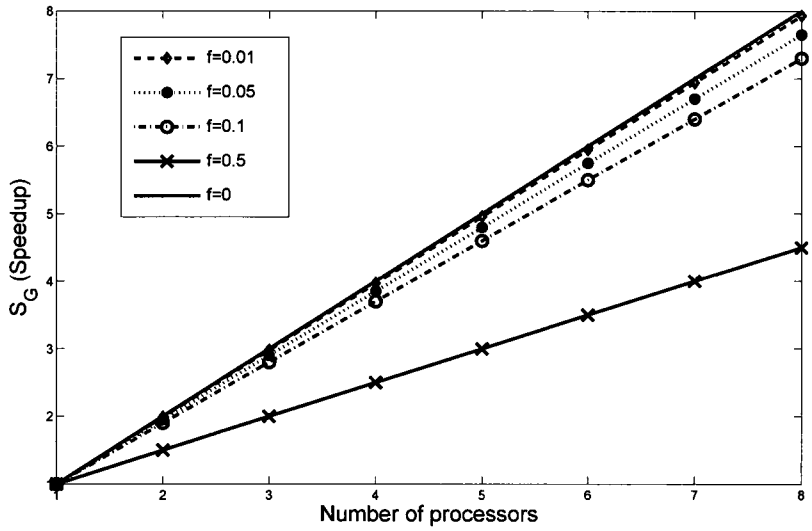

Figure 2.4: Speed Up vs Number of Processors Using Gustafson's Law

speedup model obtained from Gustafson's law is more suited to distributed memory architectures, where the memory (and the corresponding problem size) increases with the number of processors.

In the next section we discuss the different types of system architecture and their effect on the performance of the parallel algorithms.

2.4 System Architecture and Memory

System architecture and memory performance present a major bottleneck to the performance of the parallel algorithms. Proper care should be taken while designing a parallel algorithm such that it effectively exploits the system and memory structures.

In most of the contemporary parallel computers, CPUs operate considerably faster than the main memory to which they are attached. In the early years of parallel systems development, the CPUs generally ran slower than their memory. But after the performance lines crossed (i.e. CPUs became much faster than the memory), CPUs have had to remain idle for considerable time while they wait for memory accesses to complete (in other words CPUs were data starved). This makes the efficient data management a major concern for parallel algorithms that perform comparatively less processing on a large set of data [4, 7].

There are three main types of memory architectures, viz. shared memory, distributed memory and hybrid distributed-shared memory systems. In the next sections

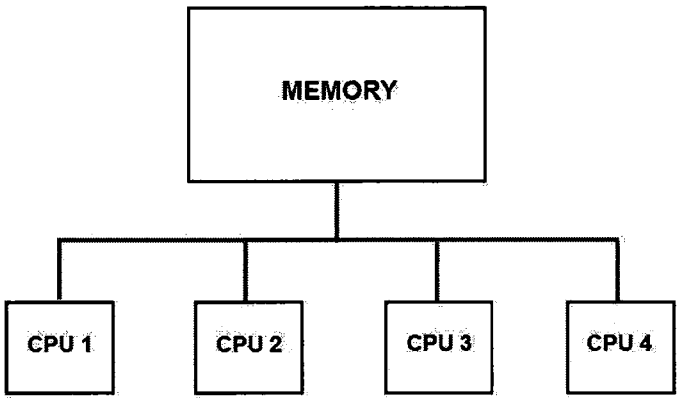

Figure 2.5: Shared Memory Architecture

we will discuss the features of these architectures.

2.4.1 Shared Memory Architecture (SMA)

In SMA, different processors in a multi-processor system share a common memory resource (such as a random access memory (RAM)). A memory bus connects the common memory to all the processors. In addition, each processor has its own registers and cache to store data. Any data changes in the shared memory are visible to all the processors which results in minimum communication between the processors. Fig. 2.5 shows a typical shared memory architecture.

The cost of communication is low in case of SMA since all the processors share common memory. Due to this designing algorithms is relatively easy on SMA and the designed algorithms are portable. The parallelization is automatic such that the

parallelism is implicit and the programmer has to devote minimum effort to distribute work among processors. Also the data sharing between the processors is fast and uniform. But, due to finite bandwidth, the shared memory traffic increases with addition of processors. Also the cost of the whole system increases as more processors are added to SMA. However, there is another disadvantage when designing algorithms for SMA. The algorithm designed must keep track of the data updated in the shared memory so that the most updated data is retrieved and there are no race conditions or multiple writes to the same memory location. The SMA exhibit limited scaling and it is comparatively difficult to achieve high levels of parallelism.

Multi-core computers are the best examples of SMA. It is to be noted that there are programming standards for various architectures which exploit their corresponding features for effective algorithm design. Open Multi-Processing (OpenMP) is the programming standard for SMA.

2.4.2 Distributed Memory Architecture (DMA)

DMA, as the name suggests, have the memory distributed over the processors which are connected by interconnect networks. There is no common shared memory as in SMA. The individual processors have their own local memories in addition to cache and registers. The communication between processors occurs over the interconnection network and the message passing is used for communication. Fig. 2.6 shows a typical

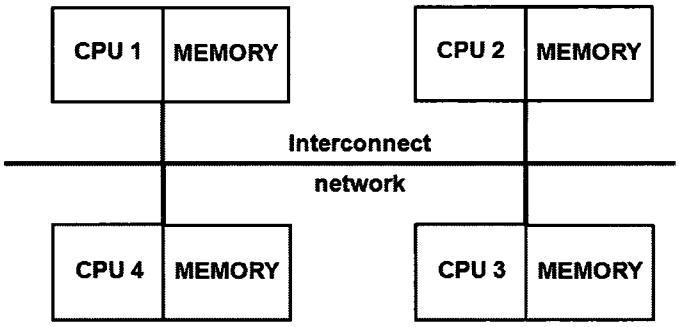

Figure 2.6: Distributed Memory Architecture

distributed memory architecture.

The advantage of DMA is its scalability. The memory available increases as the number of processors are increased. Although, the interconnection network imposes constraints on the maximum number of the processors in DMA. The number of processors that can be added to DMA (> 1000) are far more than SMA(< 500). However, DMA suffers when there is a need to communicate between processors. Since the communication has to occur over interconnect network (which is also shared by the memories), the time taken by it is significant and can have a drastic impact on the speed up obtained for DMA. The algorithm should be designed in such a way that there is minimum communication between the processors which places a serious constraint on the designer. Also, the programmer must explicitly specify parallelism for DMA. The interconnection network determines the effectiveness of the DMA because of its finite bandwidth. So to improve performance interconnection networks (often

switches) offering high bandwidth have to be used which increases the cost of the system.

Message Passing Interface (MPI) is the most widely accepted programming standard for DMA. Computer clusters are the best examples of DMA.

2.4.3 Hybrid Distributed Shared Memory Architecture

These architectures have both the distributed and shared memory. Each processor has its own memory as well as other processor's memory. This combines the advantages of both the DMA and SMA. Computer clusters again are very good examples of hybrid architectures where each node of the cluster can be a multi-core computer. Both OpenMP and MPI can be used as programming standards for these architectures. Fig. 2.7 shows a typical hybrid distributed shared memory architecture.

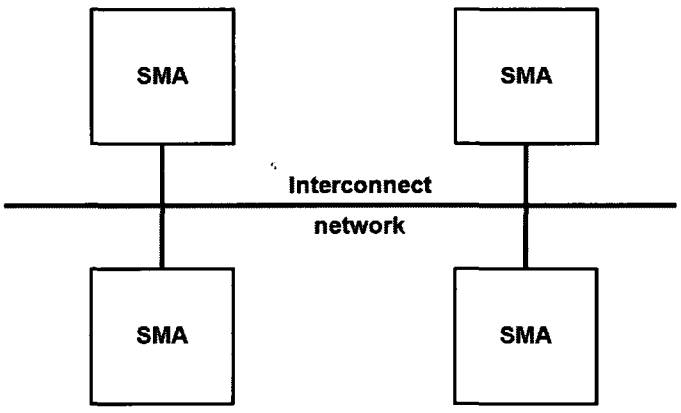

Figure 2.7: Hybrid Distributed Shared Memory Architecture

In this thesis, we will concentrate on SMA for its various advantages and availability. In the next sections, two most prominent SMA used in multi-core machines have been discussed.

2.4.4 Uniform Memory Access/Architecture

The first prominent memory architecture is the Uniform Memory Access (UMA). Intel Xeon multi-core parallel systems are based on this type of memory architecture. In UMA model all the processors share a common physical memory uniformly. This means that the memory access time is independent of which processor makes a request or which memory chip contains the transferred data. Moreover, the communication between the different processes running on different processors takes place via the same single bus which is used for memory access. Every processor in UMA also has a private cache memory. Even the peripherals can be shared between processors. Fig. 2.8 shows a typical UMA. The UMA model is suitable for general purpose and time sharing applications by multiple users. It can be used to speed up the execution of a single large program in time critical applications [4].

2.4.5 Non-Uniform Memory Access/Architecture

The second memory model is known as Cache-Coherent Non-Uniform Memory Access (ccNUMA) [7]. Cache-Coherence refers to the integrity of data stored in local caches

of a shared resource. This type of architecture can be found in AMD Opteron and Intel Itanium Nehalem and Tukwila multi-processor machines. As the name suggests the memory access is non-uniform where the memory access time depends on the memory location relative to a processor. In this memory model, each processor has a share of the total system memory, called a node. The memory has been distributed, avoiding performance hits when same processors are accessing the same memory. The inter-processor communication takes place over a separate bus. There is an additional binding on the operating system to make sure that memory allocated by the programs is in the optimal memory node. Fig. 2.9 shows a typical NUMA. The bandwidth of the ccNUMA architecture can be effectively doubled in a dual node system, as compared to UMA model for the same memory technology, by binding

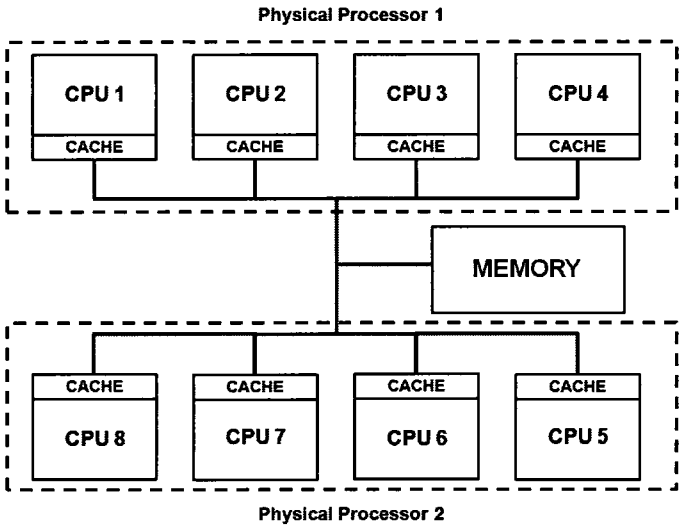

Figure 2.8: Uniform Memory Architecture

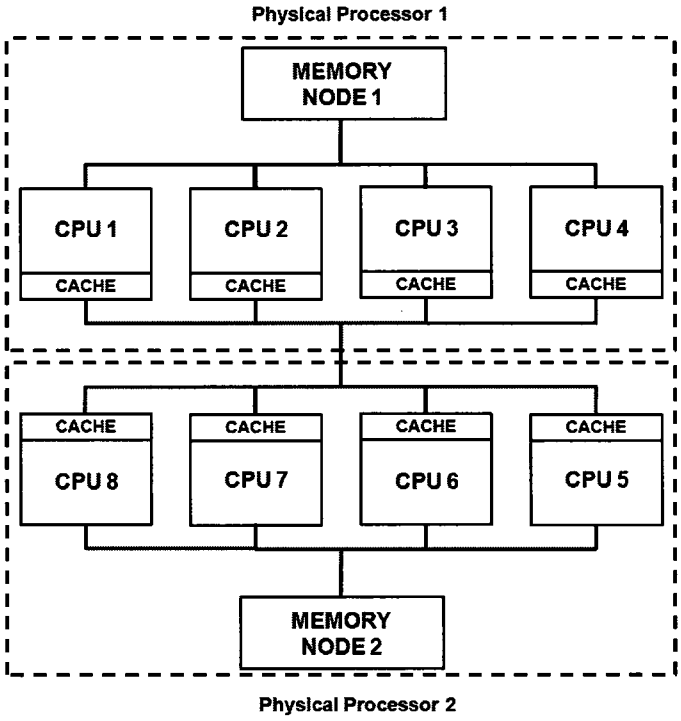

Figure 2.9: Non-Uniform Memory Architecture

the threads to specific processors to ensure memory locality. When communication between threads is minimal, a ccNUMA architecture can provide additional memory bandwidth for individual tasks to improve scaling and overall speedup [4, 7].

Chapter 3

Power Integrity in High-Speed ICs

In this Chapter, the problem of power integrity is described. The modeling of power grid networks and various techniques in literature have also been discussed.

3.1 Problem of Power Integrity in ICs

3.1.1 Problem of IC Power Distribution

To explain the problem of power distribution in ICs, we consider a basic power delivery system as shown in Fig. 3.1. It consists of a power supply, the load and the interconnect network connecting the supply to the load. The power supply is assumed to be ideal and is modeled by an ideal independent voltage source between the power (V_{DD}) and ground (V_{gnd}) levels. The interconnect network connecting the power sup-

ply to power and ground levels has an impedance of Z_p and Z_g, respectively. The load of the power distribution network has been modeled as an ideal current source, $I(t)$. The current source represents the current sinking from power to the ground levels as well as the transistor switching current.

Since the interconnect network is not ideal, there is a voltage drop across it which is given by $I \times Z$. Therefore, the voltage levels across the load is given by $V_{DD} - IZ_p$ at the power terminal and $V_{gnd} + IZ_g$ at the ground terminal. This is shown in Fig. 3.1. Hence there is a change in the supply voltage at the load terminals. This is referred to as power supply noise. The power supply noise impedes the circuit performance in many ways as explained later in this chapter. The load circuit can be designed to function properly if the supply voltage variation is kept within a certain

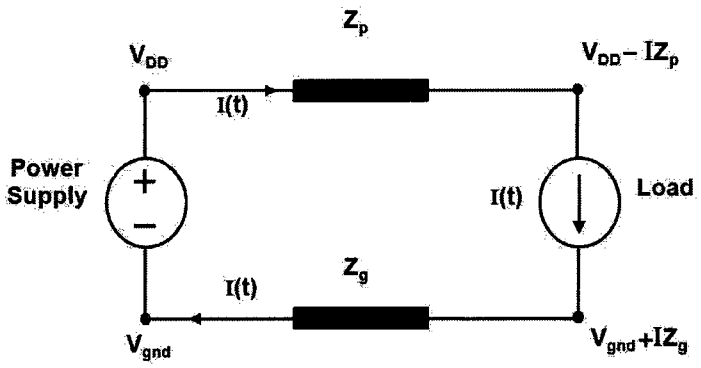

Figure 3.1: Basic Power Delivery System Consisting of Power Supply, Load and Interconnect Network

range of the nominal voltage levels. This range of variation is referred to as power noise margin. The prime objective of the designer is to design the power distribution network such that it supplies adequate current for transistors to switch and at the same time guarantees the power noise to be within the permissible noise margin.

In the next section, the effects of the power distribution noise have been elaborated.

3.2 Detrimental Effects of Power Distribution Noise

The power distribution noise has an adverse affect on the operation and performance of an integrated circuit through various mechanisms as explained in the following sections.

3.2.1 Uncertainty in Signal Delay

The propagation delay of on-chip signals depends on the power supply levels during the signal transition. For pull up networks, the source of the PMOS transistor is directly connected to the highest supply voltage. Similarly, for pull down networks, the drain of NMOS transistor is connected to ground. It is known that the drain current of the MOS is directly proportional to the voltage difference between its gate and source terminals. Therefore, when the voltage difference between the power and

ground levels is reduced (due to power distribution noise), the gate to source voltage is also reduced. The reduced gate to source voltage in turn lowers the drain current of the transistor. Hence, the transistor switches slowly thereby increasing the signal delay. On the other hand, an increase in voltage difference between power and ground levels decrease the signal delay. Thus, the effect of the power noise is to increase the signal delay uncertainty and the maximum delay of the data paths. As a result, the power supply noise limits the maximum operating frequency of an IC [8].

3.2.2 Degradation of Noise Margin

In digital communication circuits, the power and ground supply networks form the voltage reference for on-chip signals. For transmitting a high voltage state the transmitter of the communication network connects its output to the power supply network. On the other hand, for transmitting a low voltage state, the transmitter output is connected to the ground supply network. Similarly, for the receiver network, the received voltage is compared to its local power and ground voltage level. So, any variations in the supply voltage will result in inconsistency between the power and ground voltage levels at the transmitter and receiver ends of the communication network thereby degrading the noise margin for on-chip signals [8].

3.2.3 Deterioration of Gate Oxide Thickness

The process scaling has led to the improvement in the performance of MOS transistors. The current drive of the transistor increases as the thickness of the gate oxide is reduced, thereby improving speed and performance of the transistor. This reduction of oxide thickness creates the problem of electron tunneling and oxide layer reliability [8]. In addition, the power supply voltage is also limited by the maximum electric field inside the gate oxide layer. This undermines the reliability of the devices in situations where power supply variations induce voltage variation across the gate oxide above the nominal value. Hence, the variations of the power and ground supply voltages should be limited to prevent degradation in circuit operation [8]. In addition to this, power distribution noise also affects the on-chip clock jitter.

In the light of the discussion above, the design and analysis of robust power distribution networks is essential to the performance and functionality of the circuit. In the next section various components of a typical power distribution network have been discussed.

3.3 Constituents of Power Distribution Network

A PDN consists of several layers of packaging hierarchy. It consists of a switching voltage regulator module (VRM), on board PDN, on IC package PDN, on-chip PDN,

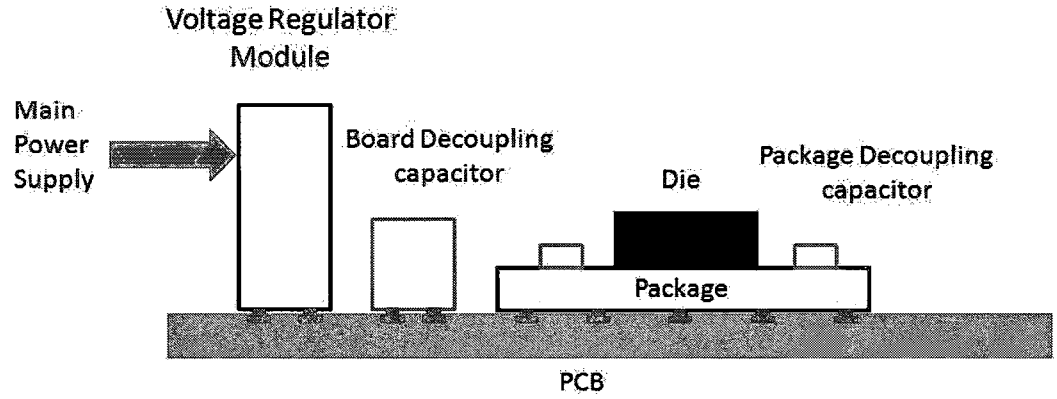

Figure 3.2: Components of a Typical Power Distribution Network

and coupling capacitors connected to these networks. The PDNs on board, package and chip form a conductive path between the power supply and load which can be logic gates, registers, flip flops, etc. Fig. 3.2 shows the various components of a typical power distribution network . The VRM converts the DC voltage level from the power supply to a voltage V_{DD} appropriate to supply to an integrated circuit. The VRM also acts as a low pass filter, filtering out the high frequency noise in the main power supply. The power and ground planes of the printed circuit board (PCB) connect the VRM to the IC package. The decoupling capacitors on the PCB act as charge reservoirs and provide the necessary charge to the fast switching transients, which switch faster than the response time of the VRM. The power and ground planes of the PCB are connected to the package power and ground networks via a ball grid array (BGA) or a pin grid array (PGA) contacts.

Similar to the board level PDN, the package power and ground networks typically consist of several metal planes. Decoupling capacitors are mounted on the package, electrically close to the die, to filter out high frequency noise and to prevent any current surges from the package PDN that may affect the power supply of the circuits on the die. The power and ground PDNs of the package are connected to the on-chip PDN through a flip-chip array of bump contacts or wire-bond contacts. Again, the decoupling capacitors are placed on the chip power and ground networks to retain low impedance at signal frequencies comparable to the switching speed of on-chip devices.

Design of on-chip power distribution networks is highly challenging because of the continual scaling of CMOS devices, significant increase circuit density and switching speeds. Additionally, increasing density of on-chip interconnects has led to a reduction in their widths, resulting in increase in the resistances of the on-chip interconnects adding to the resistance of the PDN.

The main purpose of on-chip PDN design is to provide enough power lines across the chip to avoid loss in voltage level from the bump contacts (where power is fed), to the center of the chip. Therefore, an analysis of voltage and current distribution is essential in the early stages of the design of on-chip power distribution networks. In general, an estimation of average power consumption of the chip by performing DC solution may not be sufficient and there might be a need for comprehensive analysis

of the effects of voltage and current transients produced by the rapid switching of specific functional blocks [9]. Hence, the rest of this chapter is devoted to on-chip PDN and their design and analysis.

3.4 On-Chip Power Distribution Networks

In general, power distribution networks in high performance ICs are multilayer structures. The power distribution network is laid vertically, starting from the top most metal layer, which is connected to the package, all through the interlayer vias, and to the active devices at the bottommost level. The power lines in each metal layer span the entire chip and are orthogonal to the lines in the adjacent layers. The power is supplied to the top layers from the package through interconnect contacts, called V_{DD} (or G_{nd}). The functional blocks such as logic gates, memory units, latches, etc., in the active layers act as power drains, drawing current from the PDN at the bottommost layer. Fig. 3.3 shows a typical power grid structure with 2 metal layers.

Wire Bond and Flip Chip Packages

The interconnect packages act as a connection between the PCB and IC. The behavior of on-chip PDN depends on the type of the package employed. The package interconnect that connects the PCB and IC, can be of either type, flip chip or wire

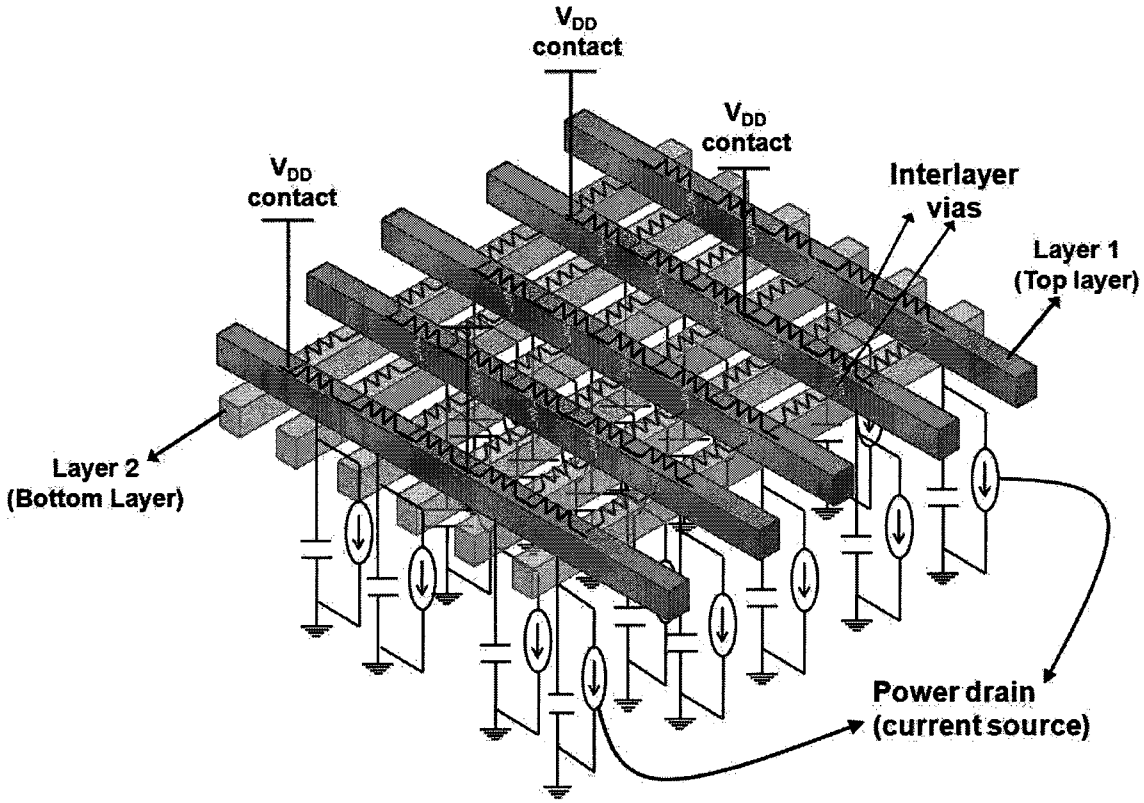

Figure 3.3: A 3-D View of a Two Layer Power Grid

bond.

In the wire bond package, V_{DD} contacts are placed along the die edges as shown in Fig. 3.4 . The IC chip is mounted face up and wires are used to connect the chip pads to the PCB. On the other hand, flip chip package (also known as Controlled Collapse Chip Connection or C4) has V_{DD} contacts distributed uniformly all over the chip. This is shown in Fig. 3.5. The chip pads on the top side of the wafer have solder deposition. These solder bumps connect the chip to the board. When the chip

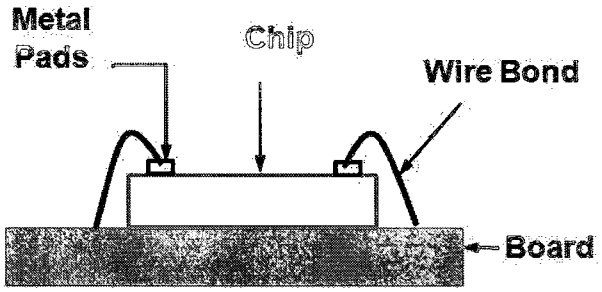

Figure 3.4: Wire Bond Package

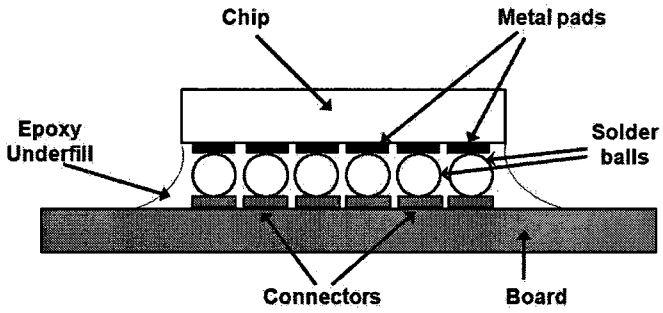

Figure 3.5: Flip Chip Package

is mounted onto the board, it is flipped over such that the top side faces down. The chip pads are aligned with the corresponding pads on the board and interconnection is completed by adding more solder.

There are advantages and disadvantages associated with both the wire bond and flip chip packages. Wire bond packages are cost effective as compared to flip chip packages. The wire bond packages are very easy to replace and can be manually

installed. Flip chip packages on the other hand are not easily replaceable. Also, the flip chip packages need a very flat surface for mounting which is difficult to arrange as the boards heat and cool. The thermal expansion of the chip must be matched to the thermal expansion of the board to avoid cracking of the connections between them.

In spite of all the above mentioned disadvantages, the flip chip packages provide superior performance compared to wire bonded packages. This is because the V_{DD} (or G_{nd}) contacts in the wire bond package are only along the edges of the on-chip PDN. This makes the active devices in the middle of the IC to be far away from the power supply. They are connected to the power supply through a highly resistive path hence the middle of the IC has a relatively lower voltage than the other parts of the IC. This creates a constraint for the designer in terms of cell placement and layout. In flip chip packages, V_{DD} (or G_{nd}) contacts are uniformly distributed over the IC. Therefore, the active devices are connected to power supply through relatively less resistive path. Hence, the voltage distribution is uniform as compared to wire bond. Further, the size of the assembly is smaller in flip chip package. The chip is mounted directly on the board resulting in short wires which significantly reduce the interconnect parasitics (like inductance), from the board thereby allowing high-speed signals and better heat conduction. Because of smaller area, flexibility, reliability and much better performance, flip chip packages are popular for lower power and miniature IC VLSI

Table 3.1: Comparison of Flip-Chip and Wire Bond Packages

Flip-Chip Package	**Wire Bond Package**
V_{DD} contacts distributed uniformly over the die	V_{DD} contacts only along the edges of the die
Package size small	Package size relatively large
Better performance	Relatively inferior performance
Low Cost	High Cost
Rugged	Less rugged
Not easily replaceable	Easily replaceable
Require a flat surface for mounting	No surface requirements

applications in recent years, compared to other packaging technologies. Hence in this thesis, flip chip based on-chip PDNs are considered for analysis. Both types of packages have been compared in Table 3.1.

3.5 Modeling of On-Chip PDN

The modeling of on-chip PDNs depends on the complexity and the accuracy desired by the designer. Generally, complex models are more accurate but at the same time,

the analysis of such complex models is computationally expensive which might not be desirable for the simulation and design of large scale PDNs.

The complexity of the model is determined by the objective of the analysis. For IR analysis, the PDN is modeled as a purely resistive network. This is because the IR analysis is performed at the preliminary phase of floor-plan based design which needs to capture only the resistive characteristics of the PDN. The capacitive and inductive circuit characteristics are not important in DC analysis, thereby significantly simplifying the model and hence the IR drop analysis.

However, when performing a transient analysis, it is essential to include the capacitive (and inductive characteristics) of the PDN. For a precise transient analysis, the capacitance of the board, package and on-chip decoupling capacitors as well as inductive properties of the PDN should be characterized with high accuracy. Since on-chip PDN are mainly capacitive and resistive in nature, and generally RC models are used to represent the power grid for various analyses. The metal segments between the interlayer vias are modeled as lumped resistors. The resistance of the vias is generally ignored to simplify the analysis. Capacitors are connected to each node in the grid to take into account the capacitive effects in the PDN. This results in a mesh type structure of resistances with capacitances connected to ground from every node as shown in Fig. 3.6. Because of the mesh type structure, on-chip PDNs are also known as power grid networks or power grids.

The power sources from the package, represented by V_{DD} contacts placed at regular intervals in a flip chip package, are modeled as ideal voltage sources. Time varying current sources are used to represent active power drains (switching devices like transistors). Fig. 3.7 shows a typical switching current waveform for such time-varying current sources. The on-chip PDN, for modern integrated circuits, comprises tens of millions of nodes and circuit elements like resistors and capacitors. This level of complexity and large problem size requires the development of highly efficient algorithm

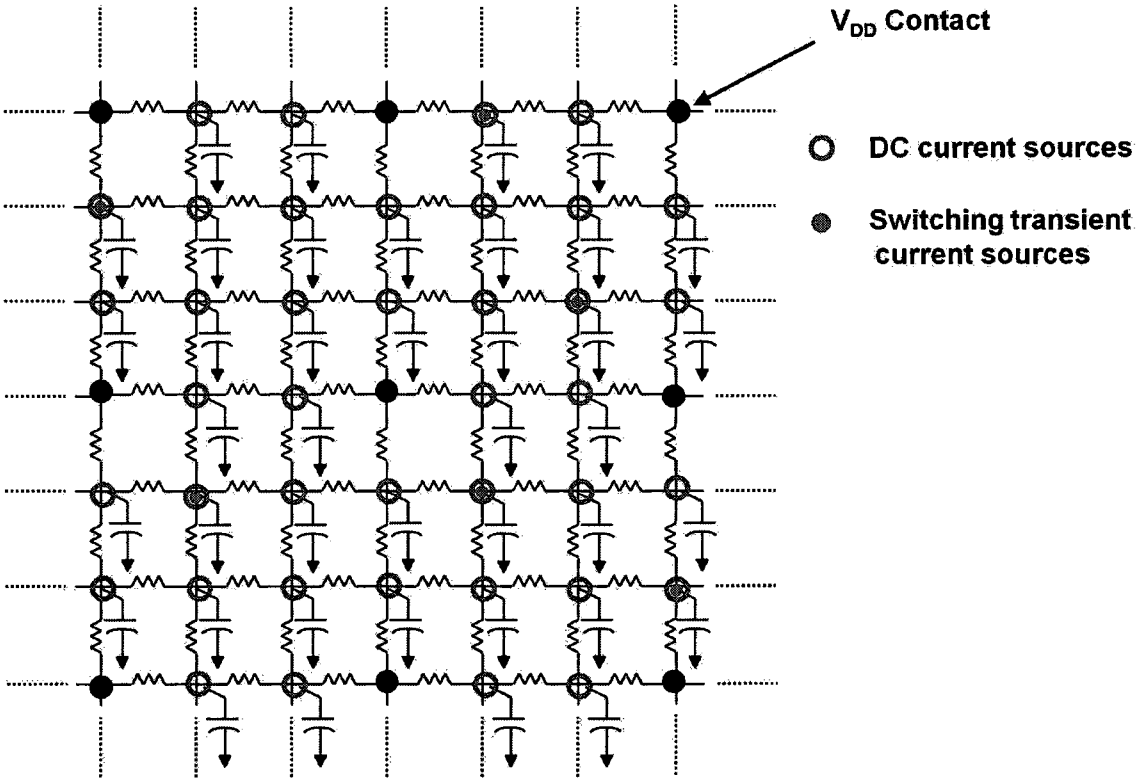

Figure 3.6: Mesh Structure of On-Chip PDNs

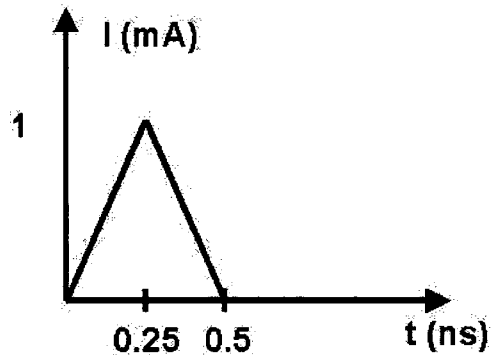

Figure 3.7: An Example Transient Current Source Waveform for Representing Active Power Drains in the PDN.

for analysis of on-chip PDNs.

In the next chapter a review of currently available power integrity analysis methods is given.

Chapter 4

Overview of Power Integrity Analysis Techniques

In this chapter, a review of power integrity analysis techniques is given. For large complex problem like power grid networks, conventional circuit simulation techniques are often inadequate. The conventional circuit simulation methods suffer from memory inefficiency, difficulty of parallelization and sometimes inability to converge to the correct solution. Various conventional circuit simulation techniques have been presented in Section 4.1. To address the difficulties with conventional techniques, waveform relaxation (WR) techniques were presented for the analysis of power grid networks. Section 4.2 provides an overview of the theory and mathematical formulation for WR techniques. Section 4.3 discusses WR techniques as relevant to the

application of power grid networks.

4.1 Conventional Techniques for Analysis of Power Distribution Networks

The modified nodal analysis (MNA) equations describing the behavior of power grid networks are as follows:

$$\boldsymbol{G}\boldsymbol{X}(t) + \boldsymbol{C}\dot{\boldsymbol{X}}(t) = \boldsymbol{b}\boldsymbol{u}(t), \tag{4.1}$$

where $\boldsymbol{X}(t)$ represents the vector of voltages on the nodes of the circuit, $\boldsymbol{G}$ and $\boldsymbol{C}$ are the conductance and capacitance matrices of the circuit, $\boldsymbol{u}(t)$ is the vector of inputs to the circuit and $\boldsymbol{b}$ is a selection matrix. The backward Euler (BE) numerical integration method can be used for solving (4.1). Applying BE to (4.1) with a step size (h) we get

$$(\boldsymbol{G} + \boldsymbol{C}/h)\boldsymbol{X}(t) = \boldsymbol{b}\boldsymbol{u}(t) + (\boldsymbol{C}/h)\boldsymbol{X}(t-h). \tag{4.2}$$

The equation (4.2) can be further simplified to

$$\boldsymbol{A}\boldsymbol{X}(t) = \boldsymbol{B}, \tag{4.3}$$

where $\boldsymbol{A}=(\boldsymbol{G} + \boldsymbol{C}/h)$ and $\boldsymbol{B}=\boldsymbol{b}\boldsymbol{u}(t) + (\boldsymbol{C}/h)\boldsymbol{X}(t-h)$. Solving the above set of equations at every time point involves finding the LU factors of matrix $\boldsymbol{A}$. Since this process is very costly for large circuits, the step size h is held constant and the

inversion process is executed only once at the beginning of the simulation for all circuits (or subcircuits) in the analysis. The time step should be kept small enough to ensure the accuracy of simulation.

It is to be noted that the size of power grids for modern ICs can be millions of nodes. The size of the corresponding MNA matrices is very large. This leads to computationally expensive and time intensive transient simulations. Numerous techniques have been proposed, in the literature, to efficiently perform transient analysis [9–17]. In [18], a novel technique based on domain decomposition for parallel simulation of power grid networks was described.

Typically, all the traditional circuit simulation techniques can be grouped into two broad categories:

1. Direct techniques

2. Iterative techniques

The direct methods rely on factoring the matrix $\boldsymbol{A}$ in equation (4.3). Once the matrix decomposition is performed, the solution at each simulation time step is obtained by forward backward substitutions. Alternatively, iterative methods can be used to obtain solution through a series of successive approximations. Direct methods offer better speed ups compared to the iterative methods, which are slow to converge due to smooth spatial variations in voltage, for power grid networks. However, direct

methods need large memory to store the LU factors of the matrices and are difficult to parallelize. On the other hand, iterative methods are more efficient in solving large systems with limited memory resources and can be easily parallelized. In the following sections, the direct and iterative methods, available in the literature, have been reviewed.

4.1.1 Direct Techniques

Direct simulation techniques involve the factorization the MNA matrices. The simplest direct method is the inversion of the MNA matrices to find the solution. But inverting a large matrix is computationally costly ($O(N^3)$). So to avoid matrix inversion, matrix factorization is used. Typically, lower upper (LU) decomposition method is employed to factorize the MNA matrices. This is followed by forward backward (FB) substitutions to obtain the solution. Direct simulation method is highly accurate and fast. There is no approximation involved and the solution is obtained in one or two steps. However, direct techniques consume a lot of memory. Hence, direct methods are not suitable for large problems. It is also difficult to parallelize direct simulation methods. Next, some of the direct techniques used for power integrity analysis have been discussed.

4.1.1.1 Multigrid Method

The efficient analysis of power grid networks can be performed by multi-grid methods proposed in [9,13]. In multi-grid methods, the power distribution network is coarsened by removing nodes in the grid. Then this reduced network is simulated for the given time interval. The solution so obtained is mapped back to the original grid by using algebraic multigrid (AMG) interpolation method. There can be many levels of reduction, depending on the accuracy and efficiency desired, for a given power grid. The reduction results in a smaller problem to be solved, which significantly reduces the simulation time. However, the interpolation of the results of the coarse grid to the fine grid produces unpredictable errors in the solution which have no correlation to the reduction of the grid. Hence, the power grid is only solved approximately.

4.1.1.2 Hierarchical Analysis

The analysis and optimization of PDN as a global partition feeding multiple local partitions can be performed by specifically formulated hierarchical analysis method [10].The process of hierarchical analysis proceeds in three stages.

- First, the entire power grid is divided into a number of local partitions.
- Next, each local partition is replaced with an equivalent circuit model obtained from the passive reduced-order interconnect macromodeling algorithm (PRIMA).

- Then, the individual partitions are simulated independently.
- Finally, these macromodels are input as multi-port circuit elements at the corresponding interface nodes in the global partition, and the resulting reduced grid is simulated.

A number of approximations are performed, during the macromodeling of the local partitions, to preserve sparsity and efficiency. Therefore, this method sacrifices a certain degree of accuracy to achieve lower run time and less memory requirements.

4.1.2 Iterative Techniques

On the other hand, iterative techniques do not involve factorization of the MNA matrices. These methods solve the problem by finding successive approximations to the solution starting from an initial guess. Iterative techniques are highly memory efficient and are suitable to solve large problems. These methods can very easily be parallelized. However, the convergence of the iterative methods can be challenging. The convergence is problem dependent and may be slow for some problems. Thus, iterative techniques are relatively slower. In the following sections, some of the important iterative techniques used for power integrity analysis have been discussed.

4.1.2.1 Random Walks Method

In [11, 12], a method based on the classical random walks method in statistics is proposed. In this method, the traveling salesman algorithm is applied to find the voltage at each individual node. The following steps describe the method:

- We start from a node to be solved.
- From the starting node, we travel to one of the neighboring nodes at random. The probability to go from node A to neighboring node B is proportional to the conductivity among the two nodes.
- If the neighboring node is a current load, the starting node voltage is incremented or reduced according to Kirchhoff's current law at that node.
- If the neighboring node is a V_{DD} contact, the node voltage is incremented and the journey ends. A number of such experiments are carried out for the given node, and the average voltage obtained from all the experiments is calculated to be the voltage at that node.

The major disadvantage with this method is that, it is difficult to determine the number of experiments necessary to be performed to obtain the accurate voltage at a particular node. Thus, again this method is susceptible to unpredictable errors.

4.1.2.2 Successive Over-Relaxation Method

For this method, the classical successive over-relaxation (SOR) iterative technique is applied to solve for the node voltages of the power grid [15, 16]. It consists of the following steps:

- First, the power grid is divided into smaller subcircuits comprising of individual nodes (or rows).
- Next, a suitable initial guess is used to initialize the grid node voltages.
- Then, for a given iteration, each node (or row) is simulated independently assuming that the voltages of the other nodes or rows) are already known.
- The solution set of the current iteration is then updated as soon as a node (or row) has been simulated. The above process is repeated for every time point in the simulation.

For faster convergence, the successive over-relaxation method uses the information from the previous iteration to correct the results for the current iteration. This involves the process of finding an optimum weighing factor, to average the solution from the present and the previous iteration. The optimum weighing factor has an excessive CPU cost associated with its calculation. Moreover, for most power grid networks which are multilayer structures, it is difficult to define an optimum weighing factor.

As discussed in Sections 4.1.1, 4.1.2, both the direct and iterative methods have their merits and demerits. In the next section, waveform relaxation techniques are described which combine the merits of both these methods.

4.2 Waveform Relaxation Techniques for Power Grid Analysis

The waveform relaxation techniques for circuit simulation were introduced in [19–23]. Waveform relaxation techniques combine the advantages of direct and iterative methods into a single method. The main idea behind the WR method is to apply relaxation directly to the circuit MNA equations. This results in the partitioning of the circuit into independent subcircuits, which can be represented by corresponding decoupled circuit equations. These individual subcircuits can be simulated over the given time interval of interest. This provides three main advantages for using WR methods. First, since each of the subcircuits is simulated independently using conventional direct methods, the accuracy of the solution can be easily improved without significant increase in memory consumption. Second, the independent simulation of the subcircuits also provides inherent partitioning which can be exploited for parallelizing the algorithm. Third, different step-sizes and different integration methods can be used to simulate various subcircuits. Thus, accurate solution can be obtained using this

method and convergence is independent of the integration or step size employed for simulation.

It is to be noted that WR method is different from conventional iterative techniques. In the WR method, the individual sub problems are simulated over the entire time interval of interest. On the other hand, conventional iterative methods involve solving the problem at every time point. In the next section, mathematical formulation for waveform relaxation techniques, for a general dynamic system, has been described.

4.2.1 Mathematical Formulation for Waveform Relaxation Technique

Consider a dynamical system defined by the following mixed algebraic-differential equations:

$$\boldsymbol{f}(\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{u}(\boldsymbol{t})) = 0. \tag{4.4}$$

where $\boldsymbol{f}$ is a continuous function describing the dynamic behavior of the system, $\boldsymbol{x}(\boldsymbol{t})$ is the unknown vector as a function of time t, $\dot{\boldsymbol{x}}(\boldsymbol{t})$ is the time derivative of the unknown $\boldsymbol{x}(\boldsymbol{t})$, and $\boldsymbol{u}(\boldsymbol{t})$ is the vector of input variables as a function of time t. The initial conditions are given by:

$$\boldsymbol{x}(\mathbf{0}) = \boldsymbol{x}_{\mathbf{0}}, \dot{\boldsymbol{x}}(\mathbf{0}) = 0. \tag{4.5}$$

Next, we assume that the time interval of interest is $[\mathbf{0}, \boldsymbol{T}]$. The process of WR starts with partitioning the system of equations in (4.4). Let the number of partitions to be equal to n. Then equation (4.4) can be represented by the following set of m disjoint equations:

$$\begin{bmatrix} \boldsymbol{f}_1(\dot{\boldsymbol{x}}_1, \boldsymbol{x}_1, \boldsymbol{q}_1, \boldsymbol{u}(\boldsymbol{t})) \\ \boldsymbol{f}_2(\dot{\boldsymbol{x}}_2, \boldsymbol{x}_2, \boldsymbol{q}_2, \boldsymbol{u}(\boldsymbol{t})) \\ \vdots \\ \boldsymbol{f}_n(\dot{\boldsymbol{x}}_n, \boldsymbol{x}_n, \boldsymbol{q}_n, \boldsymbol{u}(\boldsymbol{t})) \end{bmatrix} = \mathbf{0}, \tag{4.6}$$

where for i=1,2,.....,n, $\boldsymbol{x_i}$ is the sub vector of the unknown variables assigned to the i^{th} partitioned subsystem, $\boldsymbol{f}_i$ is a continuous function, and $\boldsymbol{q}_i$ is the input column vector conveying coupling information from one partition to the other and is given by $\boldsymbol{q}_i = col(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_n, \dot{\boldsymbol{x}}_1, \cdots, \dot{\boldsymbol{x}}_{i-1}, \dot{\boldsymbol{x}}_{i+1}, \cdots, \dot{\boldsymbol{x}}_n)$. d_i is referred to as the decoupling vector of the system.

The relaxation process is an iterative procedure. The two main types of relaxation processes employed for WR techniques are Gauss-Jacobi (GJ) and Gauss-Seidel (GS) relaxation. For the GJ relaxation, all the waveforms of the decoupling vectors (for the partitioned subsystems) are updated, from the results of the given iteration, at the beginning of the next iteration. On the other hand, for the GS relaxation, the waveforms obtained by solving one partitioned subsystem are used to update the waveforms of the decoupling vectors of the other subsystems immediately. The gen-

eral WR algorithm can be described by the following steps:

Step I: Partition the system represented by equation (4.4) into n partitions represented by equation (4.6)

Step II: Assuming r to be iteration count, set $r = 1$ and initialize the waveforms for the unknown vectors $(\boldsymbol{x}^0(t); t \in [0, T])$ such that $\boldsymbol{x}^0(0) = \boldsymbol{x}_0$.

Step III: For each partition $i = 1, 2, ..., n$, set

$$\boldsymbol{q}_i^r = col(\boldsymbol{x}_1^{r-1}, \cdots, \boldsymbol{x}_{i-1}^{r-1}, \boldsymbol{x}_{i+1}^{r-1}, \cdots, \boldsymbol{x}_n^{r-1}, \dot{\boldsymbol{x}}_1^{r-1}, \cdots, \dot{\boldsymbol{x}}_{i-1}^{r-1}, \dot{\boldsymbol{x}}_{i+1}^{r-1}, \cdots, \dot{\boldsymbol{x}}_n^{r-1}) \tag{4.7}$$

for GJ relaxation, or

$$\boldsymbol{q}_i^r = col(\boldsymbol{x}_1^r, \cdots, \boldsymbol{x}_{i-1}^r, \boldsymbol{x}_{i+1}^{r-1}, \cdots, \boldsymbol{x}_n^{r-1}, \dot{\boldsymbol{x}}_1^r, \cdots, \dot{\boldsymbol{x}}_{i-1}^r, \dot{\boldsymbol{x}}_{i+1}^{r-1}, \cdots, \dot{\boldsymbol{x}}_n^{r-1}) \tag{4.8}$$

for GS relaxation.

Step IV: Solve for $(\boldsymbol{x}_i^r(t); t \in [0, T])$ using

$$\boldsymbol{F}_i(\dot{\boldsymbol{x}}_i^r, \boldsymbol{x}_i^r, \boldsymbol{q}_i^r, \boldsymbol{u}) = \boldsymbol{0}. \tag{4.9}$$

Step V: If solution converged within the user specified error tolerance then exit else $r = r + 1$ and go to Step III.

Fig. 4.1 shows the Gauss-Jacobi (GJ) and Gauss-Seidel (GS) waveform relaxation (WR) techniques pictorially.

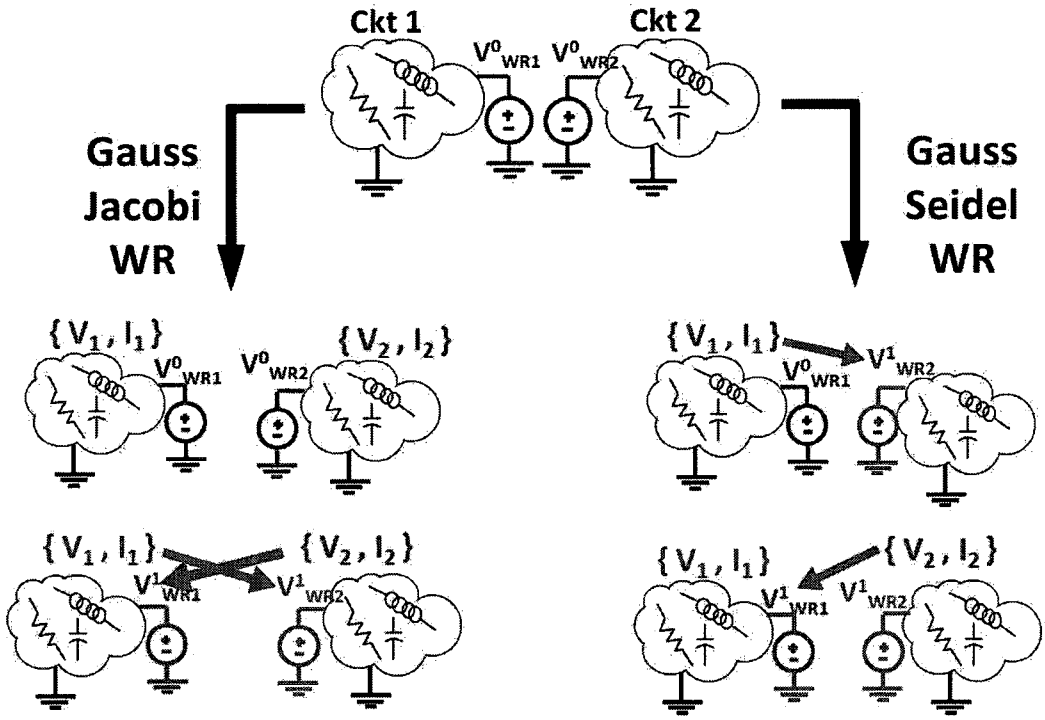

Figure 4.1: Gauss-Jacobi (GJ) and Gauss-Seidel (GS) Waveform Relaxation (WR) Techniques

4.2.2 The WR Method for a MNA Based System of Circuits

In this section, a simple circuit is used to illustrate the WR process as applied to the MNA formulation. Consider a simple circuit as shown in Fig. 4.2.

The MNA formulation of this circuit using Kirchhoff's current law (KCL) gives

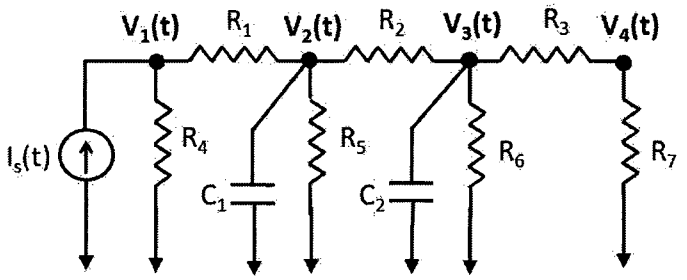

Figure 4.2: Test Circuit for Waveform Relaxation

rise to the following equations:

$$\begin{bmatrix} G_1+G_4 & -G_1 & 0 & 0 \\ -G_1 & G_1+G_5+G_2 & -G_2 & 0 \\ 0 & -G_2 & G_2+G_6+G_3 & -G_3 \\ 0 & 0 & -G_3 & G_3+G_7 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \\ v_4(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_1 & 0 & 0 \\ 0 & 0 & C_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{v}_1(t) \\ \dot{v}_2(t) \\ \dot{v}_3(t) \\ \dot{v}_4(t) \end{bmatrix} = \begin{bmatrix} I_s(t) \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{4.10}$$

where $G_j = 1/R_j$, $v_i(t)$ is the time-domain voltage waveform at the node i with an initial condition, $v_i(0) = v_{i,0}$, $\forall 1 \leq \ i \leq 4$ and $\forall 1 \leq \ j \leq 7$. Next, the circuit is partitioned into 2 parts: say one containing node 1 and node 2, while the other contains node 3 and node 4. In Fig. 4.3 , the coupling element between these two subcircuits is the resistor R_2. The coupling functions, which represent the resistive

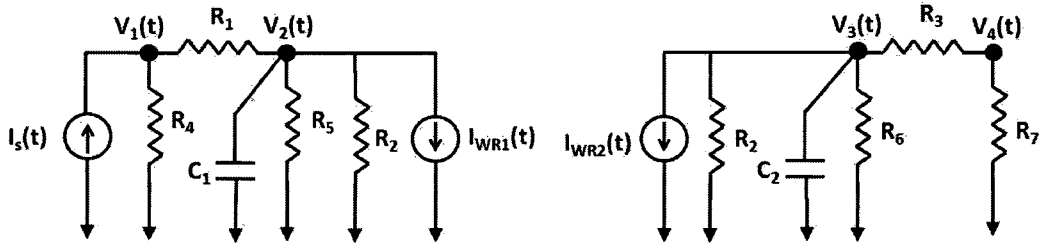

Figure 4.3: Partitioned Subcircuits with Relaxation Sources

coupling in the MNA equations for each partition, must constitute the current flowing through this resistor from node 2 to 3, written as a function of the their voltages. Hence, partitioning (4.10), the following equations are obtained.

$$\begin{bmatrix} G_1+G_4 & -G_1 & 0 & 0 \\ -G_1 & G_1+G_5+G_2 & -G_2 & 0 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \\ v_4(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{v}_1(t) \\ \dot{v}_2(t) \\ \dot{v}_3(t) \\ \dot{v}_4(t) \end{bmatrix} = \begin{bmatrix} I_s(t) \\ 0 \end{bmatrix}, \tag{4.11}$$

$$\begin{bmatrix} 0 & -G_2 & G_2+G_6+G_3 & -G_3 \\ 0 & 0 & -G_3 & G_3+G_7 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \\ v_4(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & C_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{v}_1(t) \\ \dot{v}_2(t) \\ \dot{v}_3(t) \\ \dot{v}_4(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{4.12}$$

(4.11) and (4.12) can be re-written as:

$$\begin{bmatrix} G_1+G_4 & -G_1 \\ -G_1 & G_1+G_5+G_2 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & C_1 \end{bmatrix} \begin{bmatrix} \dot{v}_1(t) \\ \dot{v}_2(t) \end{bmatrix} = \begin{bmatrix} I_s(t) \\ 0 \end{bmatrix} - \boldsymbol{I_{WR1}}(t), \quad (4.13)$$

$$\begin{bmatrix} G_6+G_3+G_2 & -G_3 \\ -G_3 & G_3+G_7 \end{bmatrix} \begin{bmatrix} v_3(t) \\ v_4(t) \end{bmatrix} + \begin{bmatrix} C_2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{v}_3(t) \\ \dot{v}_4(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \boldsymbol{I_{WR2}}(t), \quad (4.14)$$

where I_{WR1} and I_{WR2} are called waveform relaxation (WR) sources. These sources implement the functions corresponding to the decoupling vectors in (4.6). Considering r^{th} iteration of a GJ based relaxation process, these WR sources can be written as

$$\boldsymbol{I_{WR1}}(t) = \begin{bmatrix} 0 \\ -G_2 v_3(t) \end{bmatrix}, \quad (4.15)$$

$$\boldsymbol{I_{WR2}}(t) = \begin{bmatrix} -G_2 v_2(t) \\ 0 \end{bmatrix}, \quad (4.16)$$

To start the WR algorithm, these WR sources are initialized using the initial-guess waveforms, $v_i^0(t)$,$\forall 1 \leq i \leq 4$ and then updating them at the end of each iteration using the new voltage waveforms. The above formulation corresponds to the partitioned subcircuits shown in Fig. 4.3, which shows the partitioned subcircuits corresponding to (4.13) and (4.14).

4.3 Waveform Relaxation Techniques for Power Grid Networks

While using WR techniques, the power grid circuit is partitioned into smaller subcircuits. The coupling between the partitioned subcircuits is represented by relaxation sources which are attached to the boundaries of each subcircuit. Since the coupling among the partitioned power grid subcircuits is mainly resistive (which provide current paths between neighboring subcircuits), the relaxation sources can be represented by current sources. These relaxation sources are calculated using an initial guess of voltage waveforms at the boundary nodes of the subcircuits. Next, each of the small subcircuits are simulated for a given time interval of interest. In this step, the merits of the state-of-the-art fast direct solvers can be exploited. All relaxation sources are then updated according to the new values of voltages and currents in the subcircuits. The subcircuits are simulated again and this process is repeated until convergence is achieved. Further, block row partitioning (BRP) and improved method to compute initial guess waveforms are employed for fast convergence of the relaxation process. A detailed review of these techniques for power integrity analysis is given in the following sections.

4.3.1 Block Row Partitioning

In the case of using WR techniques for power grid analysis, a partitioning of the power grid network is critical for the convergence of relaxation process in WR techniques. To address the issue of partitioning, block row partitioning scheme was developed. In this scheme, the given power grid is partitioned into subcircuits consisting of blocks of rows enclosed by VDD contacts as shown in Fig. 4.4.

The BRP scheme exploits the localized voltage transients in the power grids and identifies subcircuits based on it and hence leads to faster convergence. A more detailed description is given in Fig. 4.5, which shows the i^{th} BRP subcircuit with relaxation sources attached (power grid assumed to have L rows and M columns).

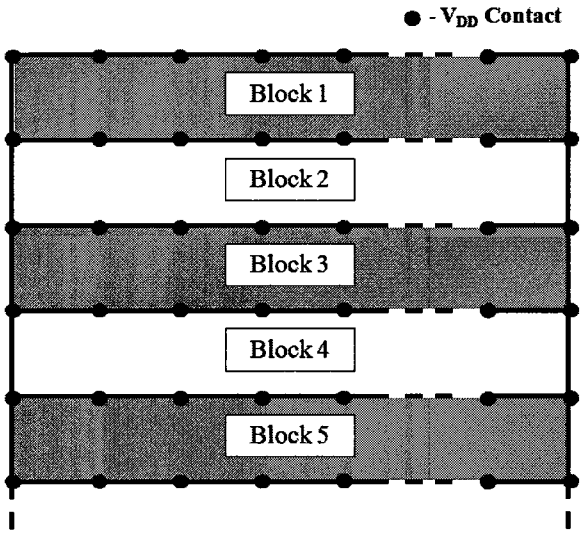

Figure 4.4: Proposed BRP Scheme for Power Grid Networks

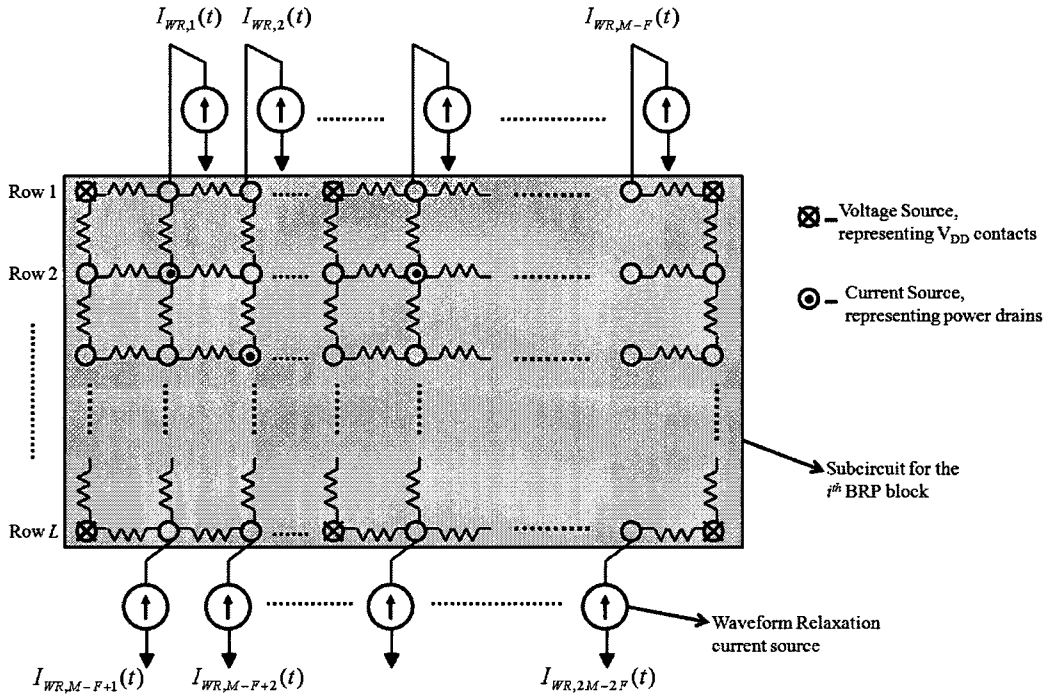

Figure 4.5: Subcircuit Description for the i^{th} Block in the BRP Scheme

The waveforms relaxation current sources can be calculated as

$$I_{WR,j}(t) = \hat{g}_j * v_{neigh,j}(t),\ \forall 1 \leq\ j \leq (2M - 2F)\,. \tag{4.17}$$

Here, F is the number of ideal voltage sources representing V_{DD} contacts, $\hat{g}_j$ is the coupling conductance term at a particular boundary node of this subcircuit and $v_{neigh,j}$ is the voltage at its (corresponding boundary node) neighboring node, in the adjoining BRP block.

The BRP scheme shows significant improvement in the convergence as compared to other partitioning schemes like single row partitioning scheme (SRP).

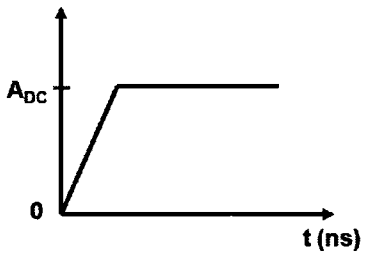

Figure 4.6: Ramped Waveform Representing DC Sources

4.3.2 Computation of Initial Guess Waveforms

The WR method in general requires that the value of the voltages at all the nodes in the circuit must be known exactly at time $t = 0$. However, for power grid networks, the calculation of the non-zero dc solution is computationally expensive. In order to overcome this difficulty, the problem of computing the DC solution is combined with WR formulation for transient analysis. Hence, the DC solution comes naturally as a result of converged WR iterations. This is achieved by modifying the various types sources in power grid networks as follows:

- **DC sources:** The DC sources in the partitioned blocks are replaced by the ramped waveforms as shown in Fig. 4.6. For DC voltage sources, $A_{DC} = V_{DD}$, and for DC current sources $A_{DC} = I_{DC}$.

- **WR sources**: The waveforms for WR current sources at the boundary nodes are also replaced by ramped waveforms similar to the DC sources as shown in Fig. 4.7. However, the corresponding amplitude A_{WR} is accurately

estimated by simulating representative source-regions for DC solution.

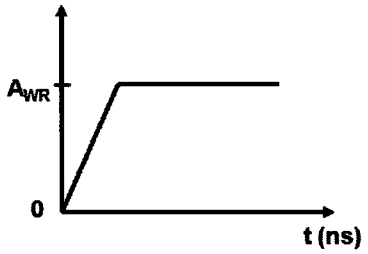

Figure 4.7: Ramped Waveform Representing the Initial-Guess for the Boundary Nodes of the Partitioned Blocks, While Computing WR Sources.

The WR based transient analysis of power grid networks provides memory efficiency as well as simulation speed-up as compared to both the direct and iterative methods. Moreover, it is easily parallelizable. An efficient parallel algorithm employing WR based techniques for power grid transient and sensitivity analysis has been presented in Chapters 5 and 6 of this thesis, respectively.

4.4 Summary

In this chapter a brief description of the theory behind the waveform relaxation was discussed and its implementation in circuit analysis based on MNA formulation was described. Also, the waveform relaxation based algorithm for power grid analysis has been presented in this chapter. Efficient partitioning schemes like BRP and

computation of initial guess waveforms has also been discussed.

Chapter 5

Proposed Parallel Transient Analysis for Power Distribution Networks

The significance of on-chip power integrity and the related challenges were presented in Chapters 3 and 4. As discussed in these chapters, conventional techniques, which are based on either direct solver methods or iterative methods, have their relative merits and disadvantages. Also it was pointed out in Chapter 4, that the WR techniques yield substantial advantages for power grid analysis.

In this chapter, a new parallel algorithm based on waveform relaxation techniques for the transient analysis of on-chip power grid circuits is developed. The new parallel

algorithm combines the advantages of both classes of methods (i.e. direct solvers and iterative solvers). Novel parallel algorithms and efficient pipelining schemes are developed to enhance the performance during transient analysis of power grids [1–3]. The speed-up and scalability of the proposed parallel algorithm is demonstrated by numerical examples.

The rest of the chapter is organized as follows. Section 5.1 develops the proposed parallel waveform relaxation based algorithm. Sections 5.2 and 5.3 present numerical results and summary respectively.

5.1 Proposed Parallel Waveform Relaxation Algorithm for Transient Analysis of Power Grid Networks

As discussed in Chapter 4, waveform relaxation has been successfully implemented for power grid analysis. The waveform relaxation based approach can be adopted for parallelization of power grid analysis because of the associated inherent partitioning. The Block-row partitions from the power grid can be simulated independently on different processors in parallel with minimum communication between them.

5.1.1 Choice of Waveform Relaxation (WR) Techniques for Parallelization

It is to be noted that, the two major WR algorithms are based on Gauss-Seidel (GS) and Gauss-Jacobi (GJ) relaxation processes [19]. However, the parallelization of the WR algorithm in the case of coupled transmission lines is generally limited to Gauss-Jacobi relaxation [24, 25]. This is because if Gauss-Seidel relaxation is used, all relaxation sources in the circuit need to be updated after each line simulation, making a parallel implementation difficult.

On the other hand, GS-WR has many merits, such as better convergence compared to the GJ-WR. In addition, GS-WR requires less memory consumption than GJ-WR. This is because, in GS-WR, the relaxation sources are computed as and when the solution is updated and hence, only one copy of the solution needs to be stored at any time. In contrast, in GJ-WR, the solution from the previous iteration is used to calculate the WR sources in the current iteration and so, two sets of the solution need to be stored at any time. Hence, a parallel GS-WR based transient analysis of power grid networks is developed exploiting the above merits of the Gauss-Seidel method.

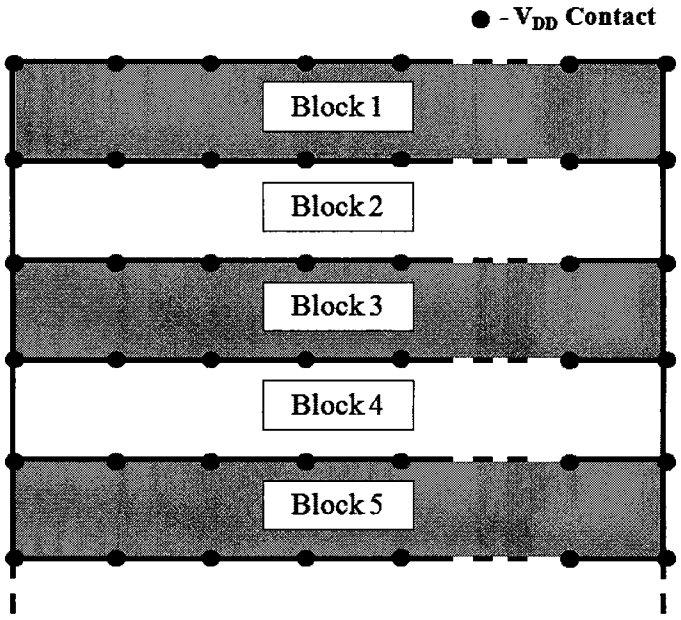

Figure 5.1: Proposed BRP Scheme for Power Grid Analysis

5.1.2 Proposed Parallel Gauss Seidel Waveform Relaxation for Power Grids

In the case of power grids, partitioning the power grid circuit into subcircuits consisting of blocks of rows (BRP scheme) as discussed in Chapter 4, gives us the opportunity to parallelize even the GS-WR algorithm. This is because, the coupling of individual subcircuits is limited to the immediate neighboring subcircuits. For this, the order of subcircuits to be simulated within a given iteration can be altered to be suitable for parallel implementation.

Noting the above aspects, in the proposed algorithm, the subcircuits are divided

into odd and even numbered blocks as highlighted in Fig. 5.1. If N_{BRP} is the number of blocks obatined by applying BRP scheme on the given power grid network, then the number of odd (N_O) and even (N_E) blocks is given by $N_O = N_E = N_{BRP}/2$ if N_{BRP} is even; $N_O = (N_{BRP} + 1)/2$ and $N_E = (N_{BRP} - 1)/2$ if N_{BRP} is odd. In a given iteration,

- First, each of the odd numbered blocks are simulated independently, on different processors,
- Using the resulting solutions, the relaxation sources are updated for the even numbered blocks,
- Next, the even numbered blocks are simulated independently on different processors,
- Finally, the solution from the simulation of even numbered blocks is then used to update the relaxation sources for the odd numbered blocks for the next iteration.

Note that in a given iteration, the solution of odd blocks are not affected by the solution of other odd blocks and the same holds for even blocks. This results in a reordered Gauss-Seidel relaxation process, i.e., within each phase, individual subcircuits are simulated independently and simultaneously on different processors, without affecting the Gauss-Seidel relaxation process. Pseudocode 5.1 lists the computational

steps involved in the proposed parallel GS-WR algorithm. A similar algorithm could be developed where even blocks are simulated first followed by odd blocks in a given iteration and is described in the Pseudocode 5.2

In the next section, we explore further methods to enhance the parallelization of the proposed GS-WR algorithm.

5.1.3 Enhanced Parallelization via Pipelining

The parallelization accomplished in the algorithm so far was based on physical partitioning, where each subcircuit of the power grid was simulated on separate processors (as shown in Pseudocode 5.1 and Pseudocode 5.2). Ideally, given N_p processors available, this would result in reduction of total CPU time by a factor of N_p. However, for problems where the number of physical partitions/subcircuits is not a multiple of the number of processors, the above speed-up may not be fully achievable. This can be illustrated by applying the algorithm described in Pseudocode 5.1 and Pseudocode 5.2 to a test power grid problem, given below.

Illustrative Example 1: Consider a power grid circuit, which has been partitioned into 7 subcircuits numbered $\{1, 2, 3, 4, 5, 6, 7\}$ and simulated on a system with 3 processors ($N_p = 3$). Assume that 2 iterations of GS-WR are sufficient for convergence. In the first phase of each iteration, 4 simulation tasks are executed simultaneously (the simulation of the odd subcircuits $\{1, 3, 5, 7\}$). In the second phase, 3 simulation

Step 1: Generate the initial-guess waveforms for the node voltages in the power grid using the method described in [1]. Store them in a common memory shared by N_p processors,, where N_p is the number of processors in the given system.

Step 2: Partition the power grid network into subcircuit blocks numbered 1, 2, 3 ... N_{BRP} as shown in Fig. 5.1.

Step 3: Define ϵ =tolerance error for the convergence of waveform relaxation iterations.

Step 4: Set the number of iterations, $r = 1$.

Step 5: For odd subcircuits 1, 3, 5 ... N_O, do the following in parallel using N_p processors:

1. Calculate the WR sources $I_{WR}(t)$ for the current subcircuit as described in [1], using the voltages of the boundary nodes of the neighboring subcircuits.
2. Simulate the current subcircuit for the entire time-interval of interest.
3. Update the voltages on the boundary nodes of the current subcircuit, in the shared memory.

Step 6: Repeat ***Step 5*** for even subcircuits 2, 4, 6 ... N_E.

Step 7: Compute error=norm(*the difference between the waveform at the output node from the present and previous iterations*).

Step 8:

if (error $\leq \epsilon$) exit;
else $r = r + 1$, go to ***Step 5***;
end

Pseudocode 5.1: Computational Steps for the Proposed Parallel GS-WR Algorithm where Odd Blocks are Simulated First Followed by Even Blocks in a Given Iteration.

Step 1: Generate the initial-guess waveforms for the node voltages in the power grid using the method described in [1]. Store them in a common memory shared by N_p processors, where N_p is the number of processors in the given system.

Step 2: Partition the power grid network into subcircuit blocks numbered 1, 2, 3 ...N_{BRP} as shown in Fig. 5.1.

Step 3: Define ϵ =tolerance error for the convergence of waveform relaxation iterations.

Step 4: Set the number of iterations, $r = 1$.

Step 5: For even subcircuits 2, 4, 6 ... N_E, do the following in parallel using N_p processors:

1. Calculate the WR sources $I_{WR}(t)$ for the current subcircuit as described in [1], using the voltages of the boundary nodes of the neighboring subcircuits.
2. Simulate the current subcircuit for the entire time-interval of interest.
3. Update the voltages on the boundary nodes of the current subcircuit, in the shared memory.

Step 6: Repeat ***Step 5*** for odd subcircuits 1, 3, 5 ... N_O

Step 7: Compute error=norm(*the difference between the waveform at the output node from the present and previous iterations*).

Step 8:

if (error $\leq \epsilon$) exit;
else $r = r + 1$, go to ***Step 5***;
end

Pseudocode 5.2: Computational Steps for the Proposed Parallel GS-WR Algorithm where Even Blocks are Simulated First Followed by Odd Blocks in a Given Iteration.

tasks are executed simultaneously (the simulation of the even subcircuits $\{2, 4, 6\}$). Table 5.1 shows the CPU utilization during each execution stage for this example. Here, each value in the table denotes the task being executed. The main number here refers to the subcircuit being simulated and the superscript indicates the iteration. For instance, 2^1 refers to the task of simulating the second subcircuit in the first iteration. For clarity, the simulation tasks of odd subcircuits are shown in bold. We find that, during the second and the fifth execution stages, 2 processors remain idle. Hence, the processors are clearly underutilized. Two of the processors are idle 33.33% of the time resulting in an average CPU utilization of 77.77%.

5.1.3.1 Pipelining via efficient assignment of physically partitioned blocks

To address the above issue of idle CPUs, a more efficient method of assigning simulation tasks to the available processors is developed. This is accomplished by filling the idle times in the proposed algorithm, with simulation tasks ordered such that the conditions of Gauss-Seidel relaxation process are not violated. For instance, during the second execution stage in Table 5.1, the second and third processors can be assigned the tasks 2^1 and 4^1 respectively. This is because, according to the proposed GS-WR algorithm, to allow the simulation of subcircuit 2 in a given iteration, the updated solution of the neighboring subcircuits 1 and 3 from the current iteration must be available. In our example, this is true as the subcircuits 1 and 3 have already

Table 5.1: CPU Utilization for Parallel Simulation Using Physical Partitioning for *Illustrative Example 1*

Execution Stage	CPU 1	CPU 2	CPU 3
1	$\mathbf{1^1}$	$\mathbf{3^1}$	$\mathbf{5^1}$
2	$\mathbf{7^1}$	*idle*	*idle*
3	2^1	4^1	6^1
4	$\mathbf{1^2}$	$\mathbf{3^2}$	$\mathbf{5^2}$
5	$\mathbf{7^2}$	*idle*	*idle*
6	2^2	4^2	6^2

been simulated in the first execution stage. Similar reasoning holds for subcircuit 4. Based on this, a modified physical partitioning scheme for the parallel simulation of power grids was developed, and the details are given below.

Assuming that odd-numbered subcircuits are simulated first, the conditions to be met for assigning a particular simulation task to an available processor in this new algorithm can be summarized using the following lemma.

Lemma 1: In order to simulate an odd subcircuit i ($\forall i \in (1, 3, 5....N_O)$) in a given

iteration, the following must hold:

1. The simulation of the subcircuit i must have been completed for the *previous* iteration.

2. The simulation of the neighboring even subcircuits $(i-1)$ and $(i+1)$ must have been completed for the *previous* iteration.

In order to simulate an even subcircuit j $(\forall j \in (2, 4, 6....N_E))$ in a given iteration, the following must hold:

1. The simulation of the subcircuit j must have been completed for the *previous* iteration.

2. The simulation of the neighboring odd subcircuits $(j-1)$ and $(j+1)$ must have been completed for the *current* iteration.

A similar lemma can be defined assuming even subcircuits are simulated first. Table 5.2 shows the CPU utilization during each execution stage for the same illustrative example, *Illustrative Example 1*, using the modified physical partitioning scheme. As can be seen from the table, the CPU utilization has now improved to 93.33%. It is to be noted that the task pipelining described above is possible in the case of power grid analysis due to the nature of coupling in power grid circuits and the properties of the proposed Gauss-Seidel waveform relaxation process (on the other hand, for the

case of multiconductor transmission lines with Gauss-Jacobi relaxation as suggested in [24, 25], pipelining based on physical partitioning is not possible).

5.1.3.2 Pipelining via physical and time partitioning:

To further improve the CPU utilization, individual subcircuit simulation tasks are further divided into N_{TP} time partitions. These partitions are treated as individual simulation tasks and can be scheduled more efficiently among the available processors. It must be noted that in a transient simulation, the solution of a particular time

Table 5.2: CPU Utilization for Parallel Simulation Using Modified Physical Partitioning (Shows Improved Utilization for *Illustrative Example 1* using *Lemma 1*)

Execution Stage	CPU 1	CPU 2	CPU 3
1	$\mathbf{1^1}$	$\mathbf{3^1}$	$\mathbf{5^1}$
2	$\mathbf{7^1}$	2^1	4^1
3	6^1	$\mathbf{1^2}$	$\mathbf{3^2}$
4	$\mathbf{5^2}$	$\mathbf{7^2}$	2^2
5	4^2	6^2	*idle*

partition of a given subcircuit depends upon the solution of the subcircuit at the end of the previous time partition. Hence, these two operations in a given iteration must always be in sequence. Assuming that odd-numbered subcircuits are simulated first, the conditions to be met for assigning a particular simulation task to an available processor in the proposed GS-WR algorithm using physical and time partitioning can be summarized using the following lemma.

Lemma 2: In order to simulate the time partition $T_P = k$ of an odd subcircuit i $(\forall i \in (1, 3, 5....N_O))$ in a given iteration, the following must hold:

1. The simulation of the time partition $T_P = k - 1$ of subcircuit i must have been completed in the *current* iteration.

2. The simulation of the time partition $T_P = k$ of subcircuit i must have been completed for the *previous* iteration.

3. The simulation of the time partition $T_P = k$ of neighboring even subcircuits $(i - 1)$ and $(i + 1)$ must have been completed for the *previous* iteration.

In order to simulate the time partition $T_P = k$ of an even subcircuit j $(\forall j \in (2, 4, 6....N_E))$ in a given iteration, the following must hold:

1. The simulation of the time partition $T_P = k - 1$ of subcircuit j must have been completed in the *current* iteration.

2. The simulation of the time partition $T_P = k$ of subcircuit j must have been completed for the *previous* iteration.

3. The simulation of the time partition $T_P = k$ of neighboring odd subcircuits $(j-1)$ and $(j+1)$ must have been completed for the *current* iteration.

A lemma can be defined, similar to the above, for the case when even subcircuits are simulated first.

Table 5.3 shows the CPU utilization during each execution stage for *Illustrative Example 1* in Table 5.3 using the physical and time partitioning scheme. The total time interval of simulation is divided into 3 partitions ($N_{TP} = 3$). Here, the subscript of the values in Table 5.3 denotes the time partition being simulated. For instance, $\mathbf{3_1^2}$ refers to the task of simulating the first time partition of subcircuit 3 for the second iteration. Note that the CPU utilization is now 100%. A pseudocode which implements the proposed GS-WR algorithm with the enhanced parallel scheduling (as discussed above) is given in Pseudocode 5.3.

In the proposed parallel GS-WR algorithm, the majority of the CPU time is spent in simulation of the individual subcircuits, where there is no communication between the different threads. The communication only occurs while updating the waveform relaxation sources. In all our experiments, we found that the CPU time required for updating the relaxation sources is about 0.3% of the total simulation time. Hence, the communication overhead time represents only a small fraction of the total simulation

Table 5.3: CPU Utilization for Parallel Simulation Using Physical and Time Partitioning (Shows Improved Utilization for *Illustrative Example 1* using *Lemma 2*)

Execution Stage	CPU 1	CPU 2	CPU 3
1	$\mathbf{1_1^1}$	$\mathbf{3_1^1}$	$\mathbf{5_1^1}$
2	$\mathbf{7_1^1}$	2_1^1	4_1^1
3	6_1^1	$\mathbf{1_2^1}$	$\mathbf{3_2^1}$
4	$\mathbf{5_2^1}$	$\mathbf{7_2^1}$	2_2^1
5	4_2^1	6_2^1	$\mathbf{1_3^1}$
6	$\mathbf{3_3^1}$	$\mathbf{5_3^1}$	$\mathbf{7_3^1}$
7	2_3^1	4_3^1	6_3^1
8	$\mathbf{1_1^2}$	$\mathbf{3_1^2}$	$\mathbf{5_1^2}$
9	$\mathbf{7_1^2}$	2_1^2	4_1^2
10	6_1^2	$\mathbf{1_2^2}$	$\mathbf{3_2^2}$
11	$\mathbf{5_2^2}$	$\mathbf{7_2^2}$	2_2^2
12	4_2^2	6_2^2	$\mathbf{1_3^2}$
13	$\mathbf{3_3^2}$	$\mathbf{5_3^2}$	$\mathbf{7_3^2}$
14	2_3^2	4_3^2	6_3^2

Step 1: Divide the power grid into a set of odd subcircuits S_O and even subcircuits S_E according to the BRP scheme discussed in [1]:
$S_O = \{s_o; \forall\ s_o = \texttt{subcircuits}\{1,3,5,\ldots\}\}$,
$S_E = \{s_e; \forall\ s_e = \texttt{subcircuits}\{2,4,6,\ldots\}\}$.
There are a total of N_{SO} odd subcircuits and N_{SE} even subcircuits.

Step 2: Divide the time interval of the solution into N_{TP} time partitions, where $N_{TP} = N_P$, and N_P is the number of processors in the given system.
`let` (r, s, T_P) =task of simulating the T_P^{th} time partition of subcircuit s in the r^{th} iteration.
`let` ***work_queue_O***=a queue of work tasks corresponding to odd subcircuits, initialized to NULL.
`let` ***work_queue_E***=a queue of work tasks corresponding to even subcircuits, initialized to NULL.
`let` ***solution_table***=a table of simulation results stored in a common memory location. This is initialized to the waveforms which are generated using the method given in [1].

```
for r = 1 to n_I (the number of iterations needed)
   for T_P = 1 to N_TP
      for i = 1 to N_SO
         s_o = S_O(i)
         append task (r, s_o, T_P) to the end
                of work_queue_O
      end for
      for i = 1 to N_SE
         s_e = S_E(i)
         append task (r, s_e, T_P) to the end
                of work_queue_E
      end for
   end for
end for
```

.....(contd. on next page)

```
while ((work_queue_O ∪ work_queue_E)≠ Φ)
   for p = 1 to N_P
      if (work_queue_O≠ Φ)
         let current_task =
                  head of work_queue_O
         result = verify(current_task
                     satisfies Lemma 2)
         if (result==true)
            solution_table = execute(
                  current_task, solution_table)
                  on processor p in background.
            continue to the beginning of for loop
         end if
      end if
      if (work_queue_E≠ Φ)
         let current_task =
                  head of work_queue_E
         result = verify(current_task
                     satisfies Lemma 2)
         if (result==true)
            solution_table = execute(
                  current_task, solution_table)
                  on processor p in background.
         else
            break out of for loop
         end if
      end if
   end for
   wait for processors to finish processing tasks
end while
```

Pseudocode 5.3: Computational Steps for the Proposed Parallel GS-WR Algorithm with Physical and Time Partitioning.

time.

5.2 Numerical Results

In this section, three industrial examples of power grid networks are considered to demonstrate the efficiency of the proposed parallel algorithm. The performance of the proposed parallel GS-WR algorithm is compared with a direct solver method and a purely iterative technique. In the first example, performance of the proposed parallel algorithm was tested on several medium sized problems and compared against existing methods such as direct and conventional Gauss-Seidel method. In the second and third examples, the proposed parallel algorithm was applied to two relatively large industrial power grid examples and the corresponding performance/scalability, during transient simulations, are investigated on a parallel platform.

In all the above examples, while using the proposed waveform relaxation algorithm, all the partitioned subcircuits were solved using state-of-the-art direct solver techniques in each iteration. The proposed algorithm was implemented on a C++ based linear circuit simulator (internal) with OpenMP multithreading extensions. This circuit simulator uses the backward Euler integration method for time-domain analysis. The KLU package (direct solver) [26] is used to solve the resulting system of linear equations.

Example 1:

In this example, the computational efficiency of the proposed parallel algorithm is

demonstrated on several test power grid circuits of varying sizes on a parallel multicore platform (1.60GHz Intel Xeon 8 core machine with 8 GB of RAM). In each example, the following analyses were performed.

a) First, the test power grid circuits were simulated using a purely iterative solver. For this purpose, conventional Gauss-Seidel relaxation (referred to as *CGS* method in this paper), was applied on the Kirchhoff's current law (KCL) based equations at individual nodes in the power grid circuit at each time point.

b) Next, the given circuits were simulated using a conventional direct-solver technique. The direct-solver used in our examples to solve the system of equations represented by (6.2) is the conventional LU decomposition based solver (KLU package [26]). This is referred to as the *direct-LU solver* method in this paper.

c) Finally, the same circuits were simulated using the proposed Gauss-Seidel waveform relaxation (GS-WR) algorithm (both serial and parallel), which combines the merits of both *direct* and *iterative* techniques, and the performance metrics are compared with the above analyses.

The above experiment was conducted on a parallel platform, for 6 different power grid circuits (*Ckt-1* to *Ckt-6*) of sizes varying from about 500,000 nodes to about 1.5

million nodes. We found in all our examples that about 4-5 iterations were sufficient for achieving convergence using the proposed parallel GS-WR algorithm. The CPU cost comparison of all the above three methods are given in Table 5.4. As seen from the table, the proposed parallel GS-WR algorithm, which combines the merits of both the CGS and the direct-solvers, yields significant speed-ups compared to both these techniques.

Example 2:

In this example, performance of the proposed waveform relaxation based algorithm on industrial power grid structures is demonstrated on a parallel platform. For this purpose, a large industrial power grid example, $\boldsymbol{PG}_1$, generated from a 4-metal layer PDN, was chosen. $\boldsymbol{PG}_1$ belongs to a flip-chip package based microprocessor of dimensions 10mm×10mm, built around a 65nm process [27]. The V_{DD} contacts are made at about every 300μm on the face of the chip, at the top metalization layer. The resulting circuit $\boldsymbol{PG}_1$ contained 2.5 million nodes.

All simulations in this example were run on the high performance computing virtual laboratory's (HPCVL) [28] SunFire machines. The SunFire is a symmetric multiprocessor (SMP) system based on the UltraSPARC line of processors and Solaris 10. The machine used for simulations in this example contained 72X dual-core UltraSPARC-IV+ 1.5 GHz processors with 576 GB of RAM.

Table 5.4: CPU Cost Comparison of Power Grid Analysis Using Different Methods(Example 1)

Circuit	Circuit size (# of nodes)	CPU time (sec)			
		CGS method	Direct-LU solver	Proposed GS-WR method	Proposed Parallel GS-WR method
Ckt-1	50k	358.01	80.15	21.62	**10.15**
Ckt-2	64k	537.41	116.15	29.22	**9.13**
Ckt-3	81k	590.50	162.18	37.52	**9.48**
Ckt-4	1M	755.60	230.64	47.14	**11.49**
Ckt-5	1.2M	957.28	303.89	57.96	**12.22**
Ckt-6	1.4M	1203.04	361.77	70.71	**14.17**

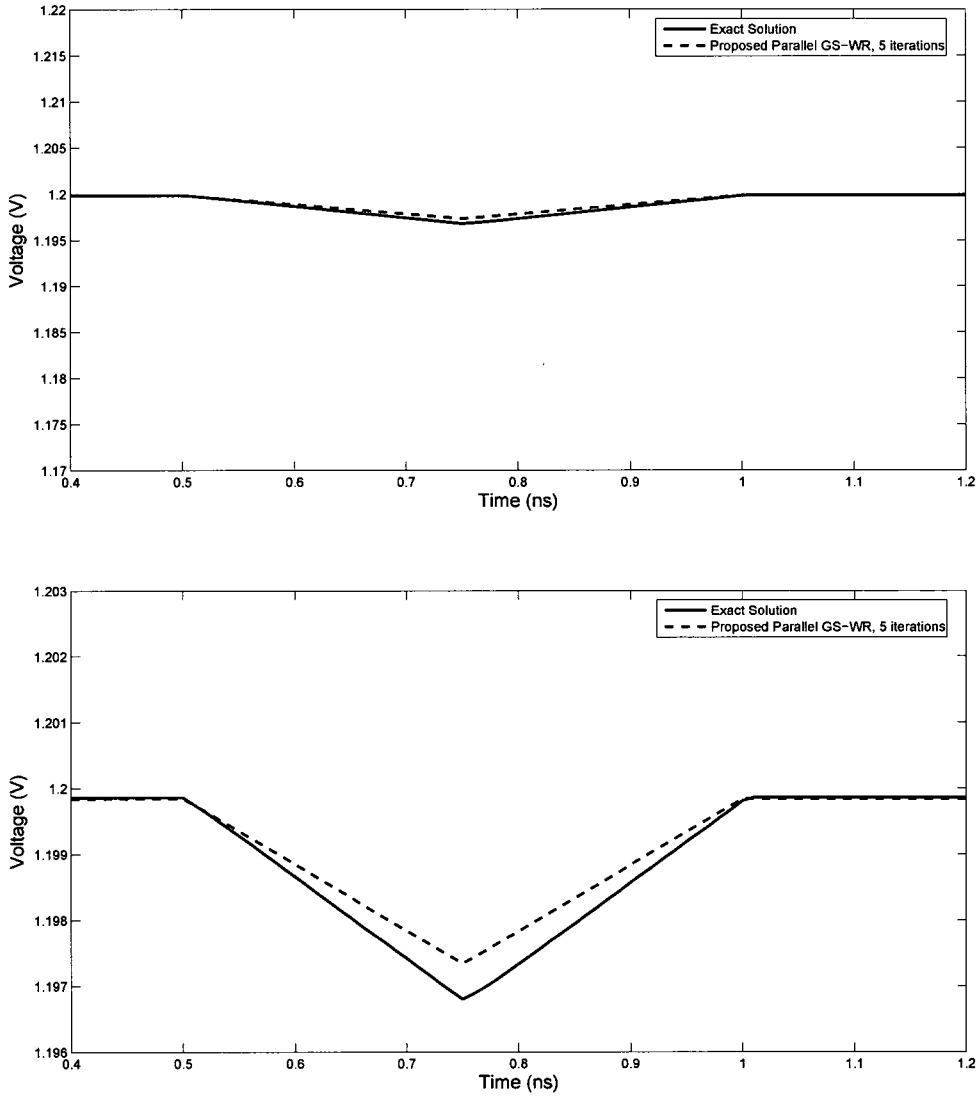

Figure 5.2: Transient Convergence Waveforms at a Sample Node in Circuit $\boldsymbol{PG}_1$ (Example 2).

These circuits were simulated for 100 time points using both the proposed parallel GS-WR algorithm, and the direct-LU solver (KLU package [26]) technique. It was found that the proposed GS-WR algorithm converges to within 0.5mV of the exact solution of these circuits, in five iterations. This is illustrated in Fig. 5.2, via the transient voltage plot at a sample node in $\boldsymbol{PG}_1$.

Next, the scalability of both these methods is investigated by running the example on different numbers of processors, and the corresponding CPU costs are given in Table 5.5. The resulting speed-ups are plotted against the number of CPUs, in Fig. 5.3.

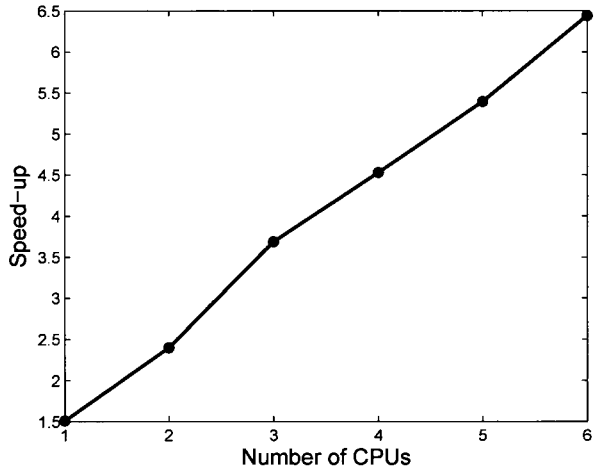

Figure 5.3: Speed-Up v/s Number of Processors for Circuit $\boldsymbol{PG}_1$ Using the Proposed Parallel GS-WR Algorithm With Respect to the Direct-LU Solver Method (Example 2).

Table 5.5: CPU Cost Comparison for Simulating Circuit $\boldsymbol{PG}_1$ on a Parallel Platform (Example 2)

Circuit $\boldsymbol{PG}_1$: # of nodes= 2.5 millions			
Proposed parallel GS-WR method		Direct-LU method	Speed-up
Number of CPUs	CPU time (min:sec)	(hr:min)	
1	42:00	1:03	**1.51**
2	26:25		**2.4**
3	17:11		**3.69**
4	13:59		**4.53**
5	11:45		**5.39**
6	9:50		**6.44**

Example 3:

The above examples were also repeated for another large industrial power grid $\boldsymbol{PG}_2$ is demonstrated. The power grid, $\boldsymbol{PG}_2$, is generated from a 5-metal layer PDN and belongs to a flip-chip package based microprocessor of dimensions 10mm×10mm, built around a 65nm process [27]. The V_{DD} contacts are made at about every 300μm on the face of the chip, at the top metalization layer. The resulting circuit for $\boldsymbol{PG}_2$ contained 7.5 million nodes. All simulations in this example were run on the high performance computing virtual laboratory's (HPCVL) [28] SunFire machines with 72X dual-core UltraSPARC-IV+ 1.5 GHz processors with 576 GB of RAM.

The resulting CPU simulation times, and the corresponding speed-ups compared to the direct-LU solver are tabulated in Table 5.6. The CPU times for the proposed parallel algorithm are plotted against the number of CPUs in Fig. 5.4, on a semilog scale. As seen, while using the proposed GS-WR parallel algorithm, the required simulation times reduced considerably with the increasing number of processors. Fig. 5.5 shows the speed-up achieved using the proposed algorithm compared to the direct solver, with varying number of processors. Fig. 5.6 shows the speed-up achieved with the proposed parallel GS-WR algorithm while using it on a single versus multiprocessor environment. As seen, the algorithm scales very well with the number of processors. In addition, Table 5.7 provides further information on the breakup of the total simulation cost (= # of CPUs × Simulation Wall Time) in terms of the total

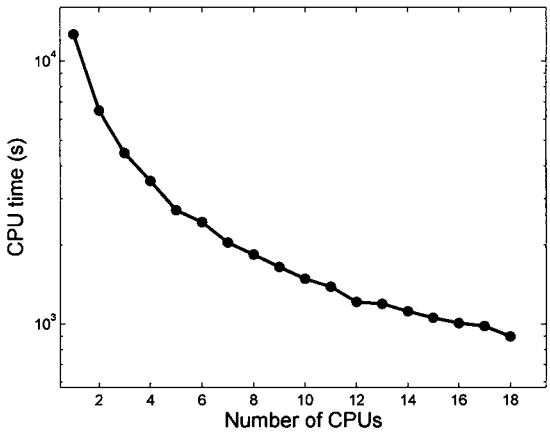

Figure 5.4: Simulation Time v/s Number of Processors for Circuit $\boldsymbol{PG}_2$ Using the Proposed Parallel GS-WR Algorithm on Different Number of CPUs (Example 3).

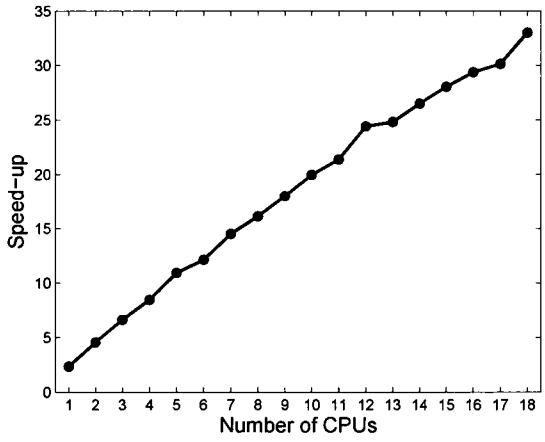

Figure 5.5: Speed-Up v/s Number of Processors for Circuit $\boldsymbol{PG}_2$ Using the Proposed Parallel GS-WR Algorithm With Respect to Direct-LU Solver Method (Example 3).

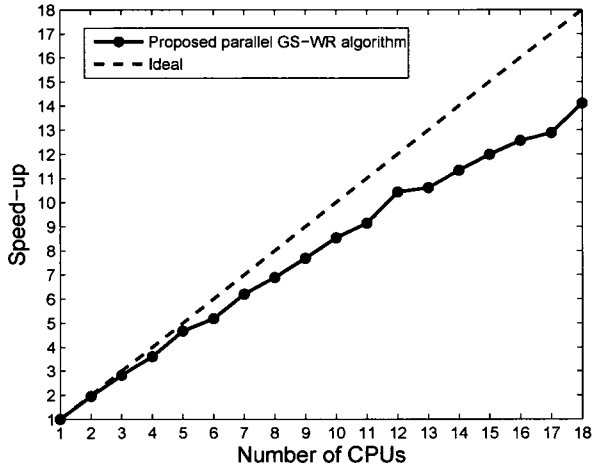

Figure 5.6: Speed-Ups for Circuit $\boldsymbol{PG}_2$ Using the Proposed Parallel GS-WR Algorithm on a Single Versus Multiprocessor Environment (Example 3).

solution cost and the total communication overhead.

5.3 Summary

In this chapter, a novel parallel algorithm for transient analysis of power distribution networks has been presented. The proposed method is based on Gauss-Seidel waveform relaxation techniques, and is hybrid in nature exploiting the advantages of both of direct and iterative solver methods. The algorithm presented proposes a parallel Gauss-Seidel waveform relaxation for power grid networks which was not possible for multi-conductor transmission networks. Different pipelining techniques have

Table 5.6: CPU Cost Comparison for Simulating Circuit $\boldsymbol{PG_2}$ on a Parallel Platform (Example 3)

Circuit $\boldsymbol{PG_2}$: # of nodes= 7.5 millions			
Proposed parallel GS-WR method		Direct-LU method	Speed-up
Number of CPUs	CPU time (hr:min:sec)	(hr:min:sec)	
1	3:31:10	8:14:07	**2.34**
3	1:14:38		**6.62**
5	0:45:12		**10.93**
6	0:40:43		**12.14**
8	0:30:39		**16.12**
10	0:24:45		**19.96**
12	0:20:14		**24.41**
16	0:16:49		**29.39**
18	0:14:58		**33.01**

Table 5.7: CPU Cost Comparison of the Total Overhead Time and Total Simulation Time for Circuit $\boldsymbol{PG_2}$ on a Parallel Platform (Example 4)

Circuit $\boldsymbol{PG_2}$: # of nodes= 7.5 millions					
Number of CPUs	Simulation Wall Time (hr:min:sec)	Solution Wall Time (hr:min:sec)	Total Simulation Time (hr:min:sec)	Total Overhead Time (hr:min:sec)	Total Overhead as % of Total Simulation Time
1	3:31:10	3:31:10	3:31:10	0:0:0	0%
3	1:14:38	1:01:54	3:43:54	0:12:44	6.03%
5	0:45:12	0:42:14	3:46:00	0:14:50	7.02%
6	0:40:43	0:35:11	4:04:18	0:33:08	15.69%
8	0:30:39	0:26:23	4:05:12	0:34:02	16.12%
10	0:24:45	0:21:07	4:07:30	0:36:20	17.21%
12	0:20:14	0:17:36	4:02:48	0:31:38	14.98%
16	0:16:49	0:13:12	4:29:04	0:57:54	27.42%
18	0:14:58	0:11:44	4:29:24	0:58:14	27.58%

been developed to improve the parallel performance and CPU utilization for the proposed parallel algorithm. As a result, the proposed parallel algorithm gives significant speed-ups compared to both direct as well as conventional iterative solvers.

Chapter 6

Parallel Time Domain Sensitivity Analysis for Power Distribution Networks

Variations in on-chip power supply can adversely affect the operation of an integrated circuit. Power supply variations due to variation in on-chip physical parameters can produce variations in the clock and data signal delays increasing the uncertainty of the timing reference signals generated on-chip and lowering the clock frequency of the circuit. It also reduces noise margins and diminishes the maximum supply voltage of the circuit, degrading the operating frequency of the circuit. Such a variation can be predicted through sensitivity analysis and hence it plays an important role in the

design and optimization of on-chip power distribution networks.

Typically, the size of power grid networks can be in the range of millions of nodes. The sheer size of these networks poses significant challenges for sensitivity analysis and design optimization. Several techniques like hierarchical symbolic relaxation and efficient sampling based sensitivity analysis [29], algebraic multigrid [30], factor analysis [31] and hot spot identification method [32] have been proposed in the literature for sensitivity analysis of power grids.

Hierarchical symbolic relaxation [29] is a direct method which embeds symbolic relaxation steps in a hierarchical fashion to reduce the circuit size. The resulting circuit can be solved by any direct method. Factor analysis [31] uses a statistical method called factor analysis and correlation to determine the power grid design variables that strongly influence voltage variation in power grid networks. Power grid sensitivity analysis using algebraic multigrid method [30], is based on another direct method by the same name in which the power grid is reduced by removing the nodes via coarsening. However in order to obtain solution of the full grid, the results from the coarse grid have to be interpolated onto the fine grid. Since, there is no direct relationship between the level of coarsening applied and the maximum error obtained in the solution, the error in the final solution is unpredictable. Factor analysis is computationally expensive and the error is unpredictable as it is a statistical method, in addition to being difficult to parallelize.

In this chapter, we extend the waveform relaxation techniques to perform sensitivity analysis of on-chip power distribution networks on a parallel platform similar to parallel transient analysis for power grid networks as described in [1]. The new algorithm has the following advantages over the existing techniques in literature:

1. It combines the advantages of both the direct and iterative methods.

2. It yields higher speed-up and scalability on parallel multi-core platforms.

The speed-up and scalability of the proposed parallel algorithm increases with the increasing problem size.

This chapter is organized as follows. Section 6.1 reviews the basics of time domain sensitivity analysis. The proposed parallel sensitivity analysis algorithm is presented in Section 6.2 and Section 6.3 provides the numerical results.

6.1 Review of Sensitivity Analysis using Direct Sensitivity Method

One of the simplest method for sensitivity analysis in time-domain is the direct sensitivity method [33]. In this method, the circuit equations are differentiated directly with respect to the varying parameter. The resulting differentiated equations result in a new circuit called the sensitivity circuit. The new sensitivity circuit has the

same topology as the original circuit. The voltages and currents in this new circuit actually represent the sensitivities of the voltages and currents with respect to the given sensitivity parameter. Hence, the solution of the sensitivity circuit provides the desired sensitivity information.

In the next section, a brief review of direct method for time-domain sensitivity analysis is presented.

6.1.1 Direct Method for Time-Domain Sensitivity Analysis

In this section, the basics of time-domain sensitivity analysis are reviewed. In the first part, the problem formulation for the direct method for sensitivity analysis is described, followed by a discussion on the various issues related to sensitivity analysis and its implementation.

Consider a power grid network described by the MNA equation

$$\boldsymbol{G}\boldsymbol{X}(t) + \boldsymbol{C}\dot{\boldsymbol{X}}(t) = \boldsymbol{b}\boldsymbol{u}(t), \tag{6.1}$$

where $\boldsymbol{X}(t)$ represents the vector of the voltages of the nodes of the circuit, $\boldsymbol{G}$ and $\boldsymbol{C}$ are the conductance and capacitance matrices of the circuit, $\boldsymbol{u}(t)$ is the vector of inputs to the circuit and $\boldsymbol{b}$ is a selection matrix. Applying BE to (6.1) with a step size of h, we get

$$(\boldsymbol{G} + \boldsymbol{C}/h)\boldsymbol{X}(t) = \boldsymbol{b}\boldsymbol{u}(t) + (\boldsymbol{C}/h)\boldsymbol{X}(t-h). \tag{6.2}$$

Following the direct sensitivity approach, differentiating equation (6.1) with respect to the sensitivity parameter α, we get

$$\boldsymbol{G}\frac{\partial \boldsymbol{X}(t)}{\partial \alpha} + \frac{\partial \boldsymbol{G}}{\partial \alpha}\boldsymbol{X}(t) + \boldsymbol{C}\frac{\partial \dot{\boldsymbol{X}}(t)}{\partial \alpha} + \frac{\partial \boldsymbol{C}}{\partial \alpha}\dot{\boldsymbol{X}}(t) = \boldsymbol{b}\frac{\partial \boldsymbol{u}(t)}{\partial \alpha}, \tag{6.3}$$

Substituting $\boldsymbol{Z}(t) = \frac{\partial \boldsymbol{X}(t)}{\partial \alpha}$ we get

$$\boldsymbol{G}\boldsymbol{Z}(t) + \boldsymbol{C}\dot{\boldsymbol{Z}}(t) = -\frac{\partial \boldsymbol{G}}{\partial \alpha}\boldsymbol{X}(t) - \frac{\partial \boldsymbol{C}}{\partial \alpha}\dot{\boldsymbol{X}}(t), \tag{6.4}$$

where $\boldsymbol{Z}(t)$ represents the vector of sensitivity of circuit node voltages with respect to the varying parameter α.

Applying BE to (6.4) with a step-size (h) we get,

$$(\boldsymbol{G} + \boldsymbol{C}/\boldsymbol{h})\boldsymbol{Z}(t) = (\boldsymbol{C}/\boldsymbol{h})\boldsymbol{Z}(t-h) - \left(\frac{\partial \boldsymbol{G}}{\partial \alpha} + \frac{1}{h}\frac{\partial \boldsymbol{C}}{\partial \alpha}\right)\boldsymbol{X}(t) - \frac{1}{h}\frac{\partial \boldsymbol{C}}{\partial \alpha}\boldsymbol{X}(t-h). \tag{6.5}$$

The circuit represented by equation (6.5) is referred to as the sensitivity circuit. It is to be noted that the sensitivity circuit has many similarities with the original circuit.

- Both the original and the sensitivity circuit share the same topology, since left-hand side (LHS) of equations (6.5) and (6.2) are identical (i.e. both the circuits have the same LHS which is $\boldsymbol{G} + \boldsymbol{C}/\boldsymbol{h}$.

- On the other hand, the right-hand side (RHS) (representing the source vectors) of both the equations are different. However, it is important to note

that the sources of the sensitivity circuit can be calculated from the solution of the original circuit (as is evident from the RHS of equation (6.5)).

In the next section, the physical parameters affecting the voltage sensitivity of power grid networks are presented.

6.1.1.1 Sensitivity Analysis for Power Grid Networks With Respect to Physical Parameters of the Power Grid

The resistance between each node of power grid networks is given by

$$R = \rho(l/w). \tag{6.6}$$

where R is the resistance between each node of power grid network, ρ is the resistivity, l is the pitch between the conductor, w is the length of the conductor.

The capacitance at each node of the power grid network is given by

$$C = \epsilon_0 \epsilon_r (A/d). \tag{6.7}$$

where C is the capacitance at each node of the power grid network, ϵ_0 is the dielectric constant of free space, ϵ_r is the relative static permittivity of the material, A is the overlap area between the conductors from different layers, and d is the distance between the layers.

To calculate the sensitivity of the node voltages with respect to the physical parameters of the power grid (i.e. the pitch and length of conductors for resistance,

overlap area and inter layer distance for the capacitance), we need to differentiate equations (6.6) and (6.7).

Differentiating equation (6.6) w.r.t. a general physical parameter, say α_{pg}, we get

$$\frac{\partial R}{\partial \alpha_{pg}} = K_{\alpha_{pg}} \left(\rho(l/w)\right), \tag{6.8}$$

Similarly, differentiating equation (6.7) w.r.t. a general physical parameter, say α_{pg}, we get

$$\frac{\partial C}{\partial \alpha_{pg}} = K_{\alpha_{pg}} \left(\epsilon_0 \epsilon_r (A/d)\right), \tag{6.9}$$

The equations (6.6) and (6.7) can be expressed as one equation as follows

$$\frac{\partial \alpha}{\partial \alpha_{pg}} = K_{\alpha_{pg}} \times \alpha. \tag{6.10}$$

where α can be either R or C, α_{pg} can be any physical parameter of the power grid, and $K_{\alpha_{pg}}$ is a constant, the value of which depends on the physical parameter being considered for sensitivity analysis and is given by

$$K_{\alpha_{pg}} = \begin{cases} \frac{1}{l} & \text{if } \alpha = \text{R and } \alpha_{pg} = \text{l}, \\ -\frac{1}{w} & \text{if } \alpha = \text{R and } \alpha_{pg} = \text{w}, \\ \frac{1}{A} & \text{if } \alpha = \text{C and } \alpha_{pg} = \text{A}, \\ -\frac{1}{d} & \text{if } \alpha = \text{C and } \alpha_{pg} = \text{d}. \end{cases} \tag{6.11}$$

The sensitivity of the power grid node voltages w.r.t. the physical parameters is

given by

$$\frac{\partial \boldsymbol{X}(t)}{\partial \alpha_{pg}} = \frac{\partial \boldsymbol{X}(t)}{\partial \alpha} \times \frac{\partial \alpha}{\partial \alpha_{pg}}. \tag{6.12}$$

where $\frac{\partial \boldsymbol{X}(t)}{\partial \alpha_{pg}}$ sensitivity of the power grid node voltages w.r.t. the given physical parameter α_{pg}, and $\frac{\partial \boldsymbol{X}(t)}{\partial \alpha}$ is the sensitivity of the power grid node voltages w.r.t. the circuit element α which can be resistance or capacitance of the power grid network.

Substituting $\boldsymbol{Z}(t) = \frac{\partial \boldsymbol{X}(t)}{\partial \alpha}$, we get

$$\frac{\partial \boldsymbol{X}(t)}{\partial \alpha_{pg}} = \boldsymbol{Z}(t) \times \frac{\partial \alpha}{\partial \alpha_{pg}}. \tag{6.13}$$

Hence, $\boldsymbol{Z}(t)$ is obtained by solving equation (6.5) and equation (6.13) gives the final sensitivity solution for the power grid node voltages w.r.t. the given physical parameter. In the next section, a brief discussion on the selection of an appropriate step size for sensitivity analysis is given.

6.1.1.2 Selection of Step Size for Sensitivity Analysis

Even though a different step size can be used for the circuit and the sensitivity equations, it makes the sensitivity analysis process highly inefficient and computationally expensive. If the step size (h) was different for both the equations, then LU factors have to be computed twice (one for the original circuit and one for the sensitivity circuit). Since computing LU factors is very expensive, using the same step size makes the process more efficient (since in this case LU factors need to be computed

only once). This is possible because both the circuits share the same circuit dynamics [34], and consequently, an appropriate selection of step size for the original circuit also provides an accurate solution of the sensitivity circuit. In this case (by using the same step size), only a few extra forward backward substitutions are required to be performed, which are computationally less expensive compared to LU factorization.

Also, the calculation of sources for sensitivity circuit from the solution of the original circuit is very simple if the same step size is used. The original circuit solution can be directly used (unaltered) to calculate the sources for sensitivity circuit (i.e. without having to interpolate the original circuit solution since both the equations (for circuit and sensitivity) are integrated with the same step size, resulting in the two circuits sharing the same time points). Pseudocode 6.1 describes the various steps that are involved while using the direct sensitivity method.

Also in Appendix A.1, a discussion of possible parallelization strategy for direct sensitivity method is given. The speed up for the direct sensitivity approach described in Appendix A.1 depends on the number of sensitivity parameters. The speed up will increase with the increasing number of parameters.Typically, parallel algorithms show appreciable speed up if the part of the problem that is requiring excessive CPU time can be parallelized [5, 6]. For the described parallelization approach in the Appendix A.1, the serial part (LU decomposition) is computationally dominant compared to the parallel part (FB substitutions). Hence, the direct differentiation

Step 1: Solve the original circuit described by equation (6.2) using the LU factors of the $(G+C/\mathrm{h})$ matrix of the entire power grid network for the time interval of interest.

Step 2: Store the transient solution of the original circuit in the memory (for access while calculating the sources for sensitivity circuit).

Step 3: Create the sensitivity circuit from the original circuit by disabling all the sources in the original circuit.

Step 4: Simulate the sensitivity circuit for the time interval of interest. At every time step:

1. Calculate the sensitivity sources from the transient solution stored in the memory in ***Step 2***, used in equation (6.3).
2. Use the LU factors previously computed in ***Step 1***, for calculating the sensitivity information.

Pseudocode 6.1: Computational Steps for the Direct Sensitivity Analysis Algorithm.

---

based parallel approach is not feasible (i.e. it would not yield significant advantages with the increasing number of processors).

In order to overcome the difficulties associated with this method, a novel algorithm is presented in the next Section, via efficient parallelization of power grid sensitivity analysis. For this purpose, WR based techniques introduced in Chapter 4 are advanced to sensitivity analysis. The WR techniques address the sensitivity analysis problem, by parallelizing the computationally dominant part of the problem, which is the LU decomposition part. This is achieved by fragmenting the problem into smaller sub-problems, whereby the smaller sub-problems are solved in parallel leading to a

high degree of parallelism.

6.2 Proposed Parallel Sensitivity Analysis of Power Grid Networks via Waveform Relaxation

In this section, the proposed parallel time-domain sensitivity analysis algorithm for power grids is presented. The new algorithm extends the waveform relaxation techniques for transient analysis [1–3] to the sensitivity analysis for the power grid problem, by appropriately partitioning the power grid circuit into smaller subcircuits. The proposed parallel algorithm for sensitivity analysis is similar to the parallel transient analysis algorithm proposed in [1–3], with some changes needed for the sensitivity analysis for power grids.

In the subsequent sections, it is assumed that the transient analysis of the given power grid network has been performed using the parallel algorithm described in Chapter 5. The transient solution has been stored in the common shared memory. The following sections explain the details of the proposed parallel algorithm. Novel partitioning schemes, an appropriate initial guess for the relaxation sources associated with the sensitivity circuits and parallelization schemes to improve CPU utilization have also been presented.

6.2.1 Proposed Partitioning Scheme Appropriate for Application of Waveform Relaxation

The partitioning of the problem plays an important role in the performance of WR based algorithms as it does in all parallel algorithms. The idea is to partition the circuit along the weak coupling between the sensitivity subcircuits. While exploiting the locality of the voltage transients in a power grid. The concept of localized behavior of voltage transients was exploited in the BRP scheme for transient analysis for power grids [1]. A brief review of BRP scheme can be found in Chapter 4.

To perform the sensitivity analysis for power grid via WR, a similar partitioning scheme has been employed. The BRP scheme proposed for transient analysis for power grid networks in [1, 3], has been extended for sensitivity analysis. As shown in equation (6.5), the right hand side depends only on the voltages and currents of the original circuit. Fig. 6.1 shows the details of the BRP scheme for the sensitivity circuit. Note that the independent sources (V_{DD} and current drains) of the original circuit have been replaced by sensitivity current sources . The rest of the circuit elements such as resistors and capacitors have not been altered. The WR sources representing the coupling between various sensitivity subcircuits behave in a similar manner to that of transient analysis. Hence, the corresponding WR sources can be computed similar to the transient analysis case (described in Chapter 5) and are shown in the Fig. 6.2.

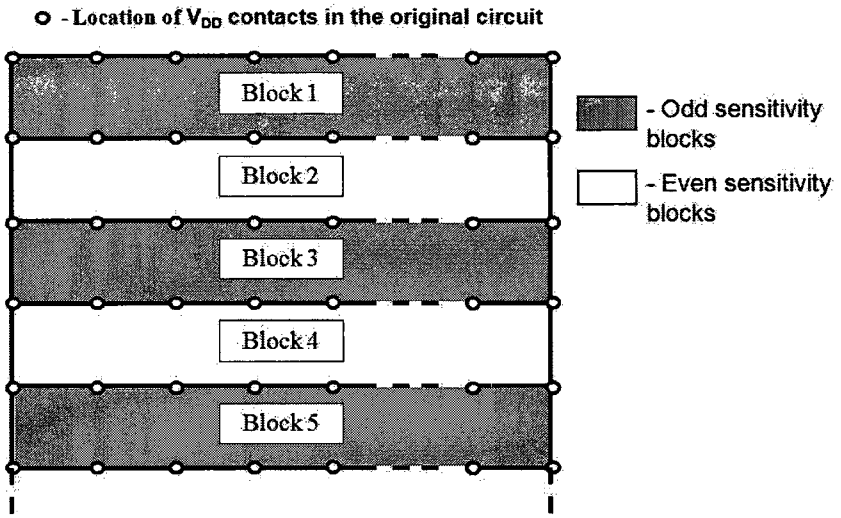

Figure 6.1: Proposed BRP Scheme for Sensitivity Analysis of Power Grid Networks

---

Fig. 6.2 also shows an i^{th} BRP sensitivity subcircuit having L rows with relaxation sources attached. Here, a power grid with M columns is considered such that, there are M resistors coupling the i^{th} BRP sensitivity subcircuit to its neighbors on either side. Thus, there are a total of $2M$ WR current sources as shown in Fig. 6.2, given by

$$I_{WRS,j}(t) = \hat{g}_j * v_{neigh,j}(t). \tag{6.14}$$

Here, $\hat{g}_j$ is the coupling conductance term at a particular boundary node of this subcircuit and $v_{neigh,j}$ is the voltage at the corresponding boundary node in the neighboring subcircuit, for all $1 \leq j \leq 2M$.

The initial waveform for the sensitivity WR sources was chosen to be zero value

(at all time points) waveform as the DC solution of the sensitivity circuit is zero. In the next section proposed parallel Guass-Seidel waveform relaxation for sensitivity analysis is presented.

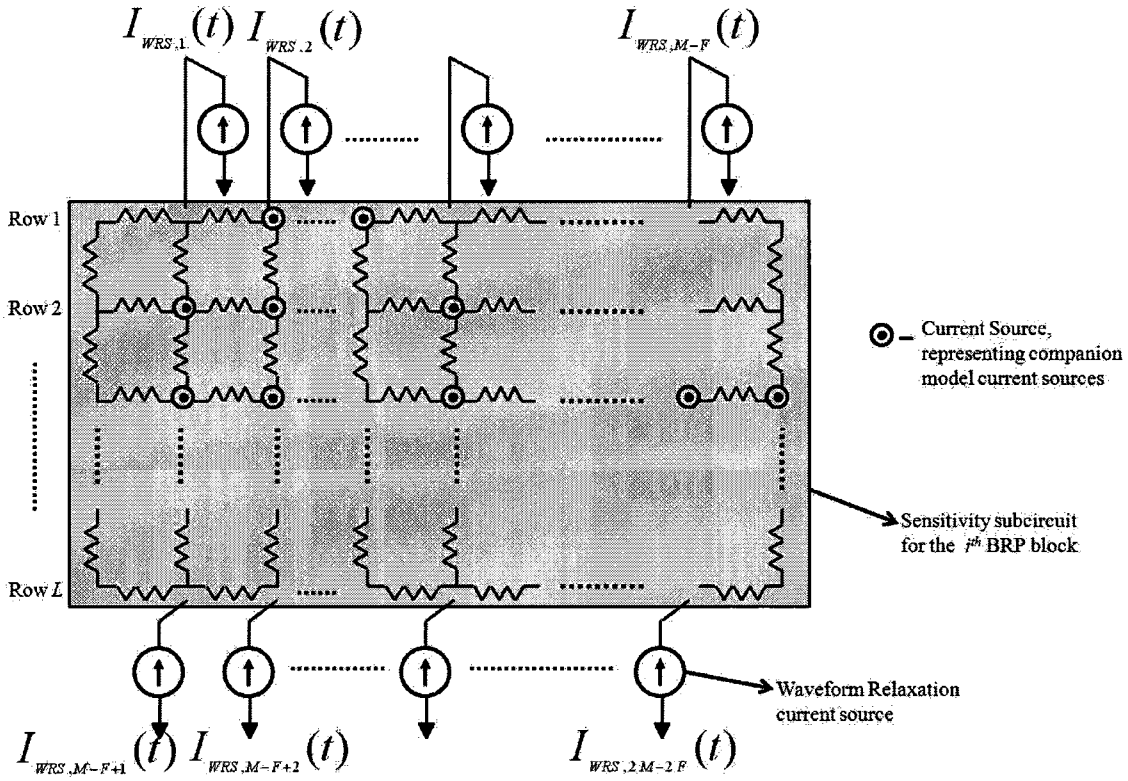

Figure 6.2: Sensitivity Subcircuit Description for the i^{th} Block in the BRP Scheme

6.2.2 Proposed Parallel Gauss Seidel Waveform Relaxation for Sensitivity Analysis of Power Grids

In this section, the parallel GS-WR proposed in Chapter 5 for transient analysis is extended for the sensitivity analysis of power grid networks. The parallelization methodology adopted for sensitivity circuits is similar to the one employed for transient analysis. The sensitivity subcircuits are divided into odd and even numbered sensitivity blocks as highlighted in Fig. 6.1. During a given iteration, odd sensitivity blocks are first simulated in parallel on different processors. Then the relaxation sources for even sensitivity blocks are updated from the solution of the odd sensitivity blocks. Next, the even sensitivity blocks are simulated in parallel. The solution from the even sensitivity blocks are then used to update relaxation sources for the odd sensitivity blocks for the next iteration. Since there is no coupling among odd sensitivity blocks (or among even sensitivity blocks), the simulation of these uncoupled blocks can be done in parallel. If $N_{S,BRP}$ is the number of sensitivity blocks obatined by applying BRP scheme on the given power grid network, then the number of odd ($N_{S,O}$) and even ($N_{S,E}$) sensitivity blocks is given by $N_{S,O} = N_{S,E} = N_{S,BRP}/2$ if $N_{S,BRP}$ is even; $N_{S,O} = (N_{S,BRP} + 1)/2$ and $N_{S,E} = (N_{S,BRP} - 1)/2$ if $N_{S,BRP}$ is odd. It is to be noted that $N_{S,BRP} = N_{BRP}$. Pseudocode 6.2 lists the computational steps involved in the proposed parallel sensitivity GS-WR algorithm. A similar algorithm could be developed where even sensitivity blocks are simulated first followed by odd

sensitivity blocks in a given iteration.

Implementation Details:

1. It is to be noted that the sensitivity analysis is started only after the transient solution has converged. This is because, the calculation of the sensitivity sources requires a converged accurate transient solution. Hence, first transient analysis is done in parallel till the solution converges. Next, sensitivity analysis is done in parallel till the sensitivity solution converges.

2. A constant step size has been used for sensitivity analysis. In [34], it was shown that same step size can be used for nominal and sensitivity circuits. Using same step size for nominal and sensitivity circuits has two advantages: a) nominal circuit LU factors can be re-used for sensitivity circuit, b) the sensitivity sources can be calculated directly from the nominal circuit solution, otherwise we need to interpolate on the nominal circuit solution to calculate the sources. However, the proposed algorithm can also be implemented with a variable step size. In this case, multiple LU decompositions will have to be performed for serial as well as parallel algorithm. For the case of serial algorithm, a certain number of LU decompositions will be done by one processor, whereas for the case of parallel algorithm the same number of LU decomposition will be done concurrently by available processors.

Step 1: Do parallel transient analysis of the power grid network using the partitioning and parallel schemes presented in [1–3].

Step 2: Set the initial-guess waveforms for the sensitivity node voltages in the power grid equal to zero. Store it in a common memory shared by N_p processors,where N_p is the number of processors in the given system. Also initialize all the WR sensitivity sources $I_{WRS}(t)$ to zero value.

Step 3: Partition the power grid network into subcircuit blocks numbered 1, 2, 3 ...$N_{S,BRP}$ as shown in Fig. 6.1.

Step 4: Define ϵ =tolerance error for the convergence of waveform relaxation iterations.

Step 5: Set the number of iterations, $sr = 1$.

Step 6: For odd sensitivity blocks 1, 3, 5 ...$N_{S,O}$, do the following in parallel using N_p processors:

1. Calculate the WR sensitivity sources $I_{WRS}(t)$ for the current subcircuit as described in (6.14), using the voltages of the boundary nodes of the neighboring subcircuits.
2. Simulate the current subcircuit for the entire time-interval of interest.
3. Update the voltages of the nodes on the boundary of the current subcircuit, in the shared memory.

Step 7: Repeat ***Step 6*** for even sensitivity blocks 2, 4, 6 ... $N_{S,E}$

Step 8: Compute error=norm(*the difference between the waveform at the output node from the present and previous iterations*).

Step 9: **if** (error $\leq \epsilon$) exit;
else $sr = sr + 1$, go to ***Step 6***;
end

Pseudocode 6.2: Summary of Computational Steps for the Proposed Parallel Sensitivity GS-WR Algorithm.

3. It is to be noted that the individual sensitivity blocks use the direct sensitivity method for sensitivity simulations as in transient analysis.

In the next section, various enhancements for the proposed parallel sensitivity GS-WR algorithm are presented.

6.2.3 Enhanced Parallelization via Pipelining

Physical partitioning is the basis of the parallelization accomplished in the algorithm so far, where each subcircuit of the power grid was simulated on separate processors (as shown in Pseudocode 6.2). The proposed GS-WR algorithm for sensitivity analysis gives a good speed up for the case where the number of processors N_p is a multiple of the number of odd/even sensitivity blocks. However, in most practical cases this might not be true. Hence, the optimal speed-up (which the parallel algorithm can exhibit) may not be fully achievable. This can be illustrated by applying the algorithm described in Pseudocode 6.2 to a test power grid problem, given below.

Illustrative Example 1: Consider a power grid circuit which has been partitioned into 7 subcircuits numbered $\{1, 2, 3, 4, 5, 6, 7\}$ and simulated on a system with 3 processors ($N_p = 3$). Assume that 2 iterations of GS-WR are sufficient for convergence. In the first phase of each iteration, 4 simulation tasks are executed (the simulation of the odd sensitivity subcircuits $\{1, 3, 5, 7\}$). In the second phase, 3 simulation tasks are executed (the simulation of the even sensitivity subcircuits $\{2, 4, 6\}$). Table 6.1

shows the CPU utilization during each execution stage for this example. Here, each value in the table denotes the task being executed. The number itself refers to the subcircuit being simulated and the superscript indicates the iteration. For instance, 2^1 refers to the task of simulating the second sensitivity subcircuit in the first iteration. For clarity, the simulation tasks of odd sensitivity subcircuits are shown in bold. We find that, during the second and the fifth execution stages, 2 processors remain idle. Hence, the processors are clearly underutilized. Two of the processors are idle 33.33% of the time giving an average overall CPU utilization of 77.77%.

6.2.3.1 Pipelining via efficient assignment of physically partitioned blocks:

To improve the speed-up or the parallel efficiency of the proposed parallel sensitivity algorithm, a more efficient method of assigning simulation tasks to the available processors is developed. In Table 6.1 there are idle times when some processors are idle and simulate no tasks. This inefficiency can be easily removed by reordering the simulation tasks such that the conditions of Gauss-Seidel relaxation process are not violated. For instance, during the second execution stage in Table 6.1, the second and third processors can be assigned the tasks 2^1 and 4^1 respectively. This is because, according to the proposed GS-WR algorithm, to allow the simulation of sensitivity subcircuit 2 in a given iteration, the updated solution of the neighboring sensitivity subcircuits 1 and 3 from the current iteration must be available. This leads to the

Table 6.1: CPU Utilization for Parallel Sensitivity Simulation using Physical Partitioning for *Illustrative Example 1*

Execution Stage	CPU 1	CPU 2	CPU 3
1	$\mathbf{1^1}$	$\mathbf{3^1}$	$\mathbf{5^1}$
2	$\mathbf{7^1}$	*idle*	*idle*
3	2^1	4^1	6^1
4	$\mathbf{1^2}$	$\mathbf{3^2}$	$\mathbf{5^2}$
5	$\mathbf{7^2}$	*idle*	*idle*
6	2^2	4^2	6^2

development of a modified physical partitioning scheme for the parallel sensitivity simulation of power grids, and the details are given below.

Assuming that odd-numbered sensitivity subcircuits are simulated first (similar to the Lemma 1 in Chapter 5), the conditions to be met for assigning a particular simulation task to an available processor in this new algorithm can be summarized using the following lemma.

Lemma A: In order to simulate an odd sensitivity subcircuit i ($\forall i \in (1, 3, 5....N_{S,O})$)

in a given iteration, the following must hold:

1. The sensitivity simulation of the subcircuit i must have been completed for the *previous* iteration.

2. The sensitivity simulation of the neighboring even subcircuits $(i-1)$ and $(i+1)$ must have been completed for the *previous* iteration.

In order to simulate an even sensitivity subcircuit j $(\forall j \in (2, 4, 6....N_{S,E}))$ in a given iteration, the following must hold:

1. The sensitivity simulation of the subcircuit j must have been completed for the *previous* iteration.

2. The sensitivity simulation of the neighboring odd subcircuits $(j-1)$ and $(j+1)$ must have been completed for the *current* iteration.

A similar lemma can be defined assuming even sensitivity subcircuits are simulated first. Table 6.2 shows the CPU utilization during each execution stage for the same illustrative example using the modified physical partitioning scheme. As can be seen from the table, the CPU utilization has now improved to 93.33%.

6.2.3.2 Pipelining via physical and time partitioning:

Until now we have explored the physical partitioning of the sensitivity analysis problem. In this subsection we divide the individual subcircuit sensitivity simulation tasks

Table 6.2: CPU Utilization for Parallel Sensitivity Simulation using Modified Physical Partitioning (Shows Improved Utilization for *Illustrative Example 1* using *Lemma A*)

Execution Stage	CPU 1	CPU 2	CPU 3
1	$\mathbf{1^1}$	$\mathbf{3^1}$	$\mathbf{5^1}$
2	$\mathbf{7^1}$	2^1	4^1
3	6^1	$\mathbf{1^2}$	$\mathbf{3^2}$
4	$\mathbf{5^2}$	$\mathbf{7^2}$	2^2
5	4^2	6^2	*idle*

into N_{TP} time partitions. This is again the extension of the same concept for parallel transient analysis for sensitivity analysis of power grids. These partitions are treated as individual simulation tasks and can be scheduled more efficiently among the available processors. The conditions to be met for assigning a particular simulation task to an available processor in the proposed sensitivity GS-WR algorithm using physical and time partitioning can be summarized using the following lemma (similar to the Lemma 2 in Chapter 5).

Lemma B: In order to simulate the time partition $T_P = k$ of an odd sensitivity

subcircuit i ($\forall i \in (1, 3, 5....N_{S,O})$) in a given iteration, the following must hold:

1. The sensitivity simulation of the time partition $T_P = k - 1$ of subcircuit i must have been completed in the *current* iteration.

2. The sensitivity simulation of the time partition $T_P = k$ of subcircuit i must have been completed for the *previous* iteration.

3. The sensitivity simulation of the time partition $T_P = k$ of neighboring even subcircuits $(i - 1)$ and $(i + 1)$ must have been completed for the *previous* iteration.

In order to simulate the time partition $T_P = k$ of an even sensitivity subcircuit j ($\forall j \in (2, 4, 6....N_{S,E})$) in a given iteration, the following must hold:

1. The sensitivity simulation of the time partition $T_P = k - 1$ of subcircuit j must have been completed in the *current* iteration.

2. The sensitivity simulation of the time partition $T_P = k$ of subcircuit j must have been completed for the *previous* iteration.

3. The sensitivity simulation of the time partition $T_P = k$ of neighboring odd subcircuits $(j - 1)$ and $(j + 1)$ must have been completed for the *current* iteration.

Table 6.3: CPU Utilization for Parallel Simulation using Physical and Time Partitioning (Shows Improved Utilization for *Illustrative Example 1* using *Lemma B*)

Execution Stage	CPU 1	CPU 2	CPU 3
1	$\mathbf{1_1^1}$	$\mathbf{3_1^1}$	$\mathbf{5_1^1}$
2	$\mathbf{7_1^1}$	2_1^1	4_1^1
3	6_1^1	$\mathbf{1_2^1}$	$\mathbf{3_2^1}$
4	$\mathbf{5_2^1}$	$\mathbf{7_2^1}$	2_2^1
5	4_2^1	6_2^1	$\mathbf{1_3^1}$
6	$\mathbf{3_3^1}$	$\mathbf{5_3^1}$	$\mathbf{7_3^1}$
7	2_3^1	4_3^1	6_3^1
8	$\mathbf{1_1^2}$	$\mathbf{3_1^2}$	$\mathbf{5_1^2}$
9	$\mathbf{7_1^2}$	2_1^2	4_1^2
10	6_1^2	$\mathbf{1_2^2}$	$\mathbf{3_2^2}$
11	$\mathbf{5_2^2}$	$\mathbf{7_2^2}$	2_2^2
12	4_2^2	6_2^2	$\mathbf{1_3^2}$
13	$\mathbf{3_3^2}$	$\mathbf{5_3^2}$	$\mathbf{7_3^2}$
14	2_3^2	4_3^2	6_3^2

Table 6.3 shows the CPU utilization during each execution stage for the example in Table 5.2 using the physical and time partitioning scheme. The total time interval of simulation is divided into 3 partitions ($N_{TP} = 3$). Here, the subscript in the values of Table 6.3 denotes the time partition being simulated. For instance, $\mathbf{3_1^2}$ refers to the task of simulating the first time partition of subcircuit 3 for the second iteration. Note that the CPU utilization is now 100%. A pseudocode which implements the proposed sensitivity GS-WR algorithm with the enhanced parallel scheduling (as discussed above) is given in Pseudocode 6.3. It is assumed that the transient solution of the circuit has been computed.

6.3 Numerical Results

In this section three numerical examples are presented to show the accuracy and efficiency of the proposed parallel sensitivity GS-WR algorithm. Results of the proposed parallel sensitivity GS-WR algorithm is compared to the traditional perturbation-based approach. Example 1 shows the convergence of WR iterations and example 2 validates the accuracy of the proposed sensitivity algorithm for a test 2 layer power grid circuit. Example 3 demonstrates the speed-up and scalability of the proposed parallel sensitivity analysis algorithm for two industrial power grid networks on parallel platform.

For all the examples, the individual subcircuits for the proposed parallel sensitivity

Step 1: Divide the power grid into a set of odd subcircuits S_O and even subcircuits S_E according to the BRP scheme discussed in [1]:
$S_O = \{s_o; \forall\ s_o = \texttt{subcircuits}\{1,3,5,\ldots\}\}$,
$S_E = \{s_e; \forall\ s_e = \texttt{subcircuits}\{2,4,6,\ldots\}\}$.
There are a total of N_{SO} odd subcircuits and N_{SE} even subcircuits.

Step 2: Divide the time interval of the solution into N_{TP} time partitions, where $N_{TP} = N_P$, and N_P is the number of processors in the given system.
`let` (r, s, T_P) =task of simulating the T_P^{th} time partition of sensitivity subcircuit s in the r^{th} iteration.
`let` ***sens_work_queue_O***= a queue of sensitivity work tasks corresponding to odd subcircuits, initialized to NULL.
`let` ***sens_work_queue_E***= a queue of sensitivity work tasks corresponding to even subcircuits, initialized to NULL.
`let` ***sens_solution_table***= a table of sensitivity simulation results stored in a common memory location. This is initialized to the zero value for all time points

```
for r = 1 to n_S (the number of sensitivity iterations needed)
   for T_P = 1 to N_TP
      for i = 1 to N_SO
         s_o = S_O(i)
         append task (r, s_o, T_P) to the end
               of sens_work_queue_O
      end for
      for i = 1 to N_SE
         s_e = S_E(i)
         append task (r, s_e, T_P) to the end
               of sens_work_queue_E
      end for
   end for
end for
```

.....(contd. on next page)

```
while ((sens_work_queue_O ∪ sens_work_queue_E)≠ Φ)
  for p = 1 to N_P
    if (sens_work_queue_O≠ Φ)
      let current_task =
            head of sens_work_queue_O
      result = verify(current_task
            satisfies Lemma 2)
      if (result==true)
        sens_solution_table = execute(
            current_task, sens_solution_table)
            on processor p in background.
        continue to the beginning of for loop
      end if
    end if
    if (sens_work_queue_E≠ Φ)
      let current_task =
            head of sens_work_queue_E
      result = verify(current_task
            satisfies Lemma 2)
      if (result==true)
        sens_solution_table = execute(
            current_task, sens_solution_table)
            on processor p in background.
      else
        break out of for loop
      end if
    end if
  end for
  wait for processors to finish processing tasks
end while
```

Pseudocode 6.3: Computational Steps for the Proposed Sensitivity Parallel GS-WR Algorithm with Physical and Time Partitioning.

GS-WR algorithm were simulated using state-of-the-art direct solver techniques in each iteration. A C++ based circuit simulator was developed to implement the proposed parallel algorithm. The simulator uses the BE method for integration of

the differential equations. An OpenMP based parallel platform was used for the implementation of the proposed algorithm. The KLU package (a direct solver) [26] is used to solve the resulting system of linear equations.

Example 1:

In this example, the convergence of proposed parallel GS-WR algorithm is shown. For this purpose, a two layer test power grid circuit is constructed. The resulting power grid structure contains 60k nodes. The sensitivity parameter chosen is the overlap area of the conductors in metal layers (for details see equations (6.10) to (6.13)). All the simulations were run on a 1.60GHz Intel Xeon dual quad-core machine with 8 GB of RAM running Gentoo Linux. The result is compared with perturbation with respect to the same parameter.

Fig. 6.3 shows the waveforms for the proposed GS-WR algorithm. It can be seen that the proposed algorithm takes 10 WR iterations for convergence of the sensitivity solution (in addition to 10 WR iterations for convergence of the transient solution). This demonstrates the convergence of the proposed GS-WR sensitivity algorithm.

Example 2:

In this example, the convergence and the accuracy of the proposed sensitivity GS-WR algorithm is presented. The test power grid circuit taken in this example is same as Example 1. Traditional perturbation technique is used to compare the accuracy of the proposed algorithm. The sensitivity parameters chosen for this analysis are the

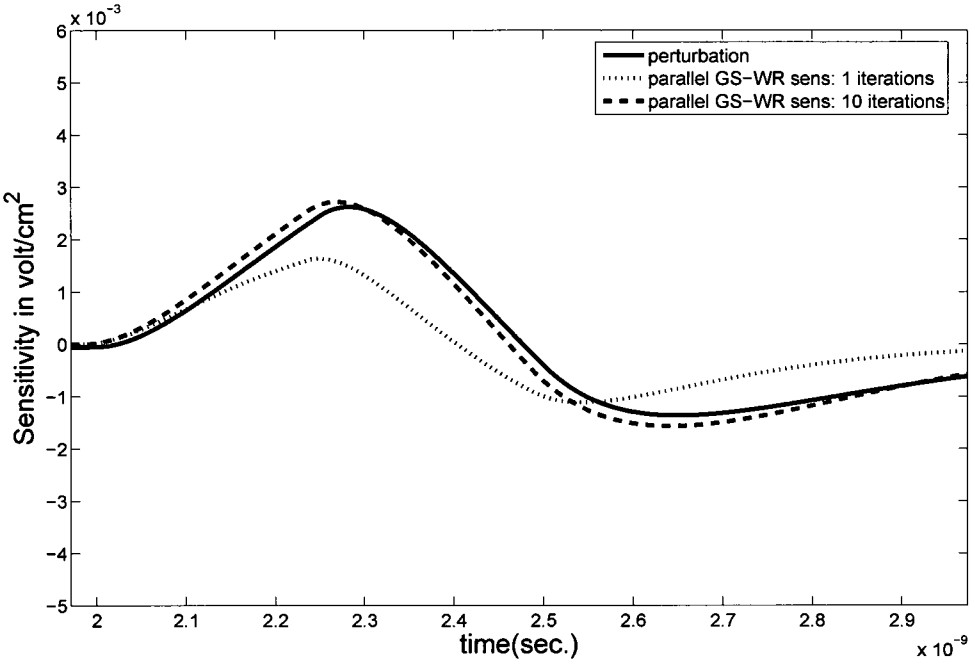

Figure 6.3: Convergence of WR Iterations for the Proposed GS-WR Based Sensitivity Analysis Algorithm

physical parameters of the power grid like the pitch and the width of the metal layers or the overlap area for the capacitance of the grid (for details see equations (6.10) to (6.13)). In all the simulations 20 WR iterations were sufficient to achieve convergence within 5% of the exact waveforms obtained from perturbation. The simulations were run on a 1.60GHz Intel Xeon dual quad-core machine with 8 GB of RAM running Gentoo Linux.

Fig. 6.4 shows the comparison between perturbation and the proposed GS-WR

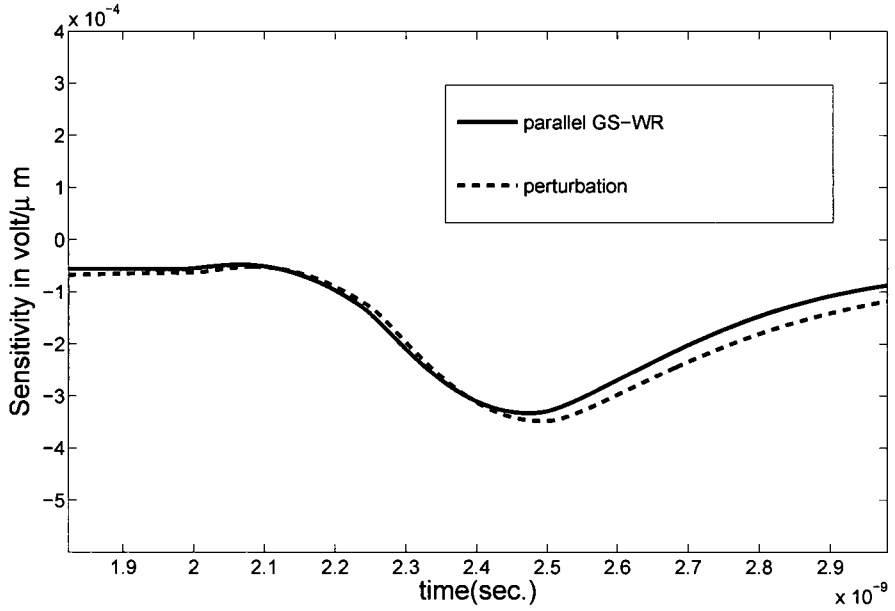

Figure 6.4: Sensitivity Convergence Waveform v/s the Variations in the Pitch of Metal Layer 1 (Example 2)

based sensitivity algorithm for the pitch of the conductors in metal layer 1 as the sensitivity parameter for the two layer power grid example under consideration. As can be seen from the figure, the proposed GS-WR based sensitivity algorithm waveform converges to the perturbation waveforms for the given sensitivity parameter.

Fig. 6.5 shows the waveforms for the width of the conductors as the sensitivity parameter and Fig. 6.6 demonstrates the convergence waveforms for the case when the overlap area for the capacitance of the power grid is selected as the sensitivity parameter.

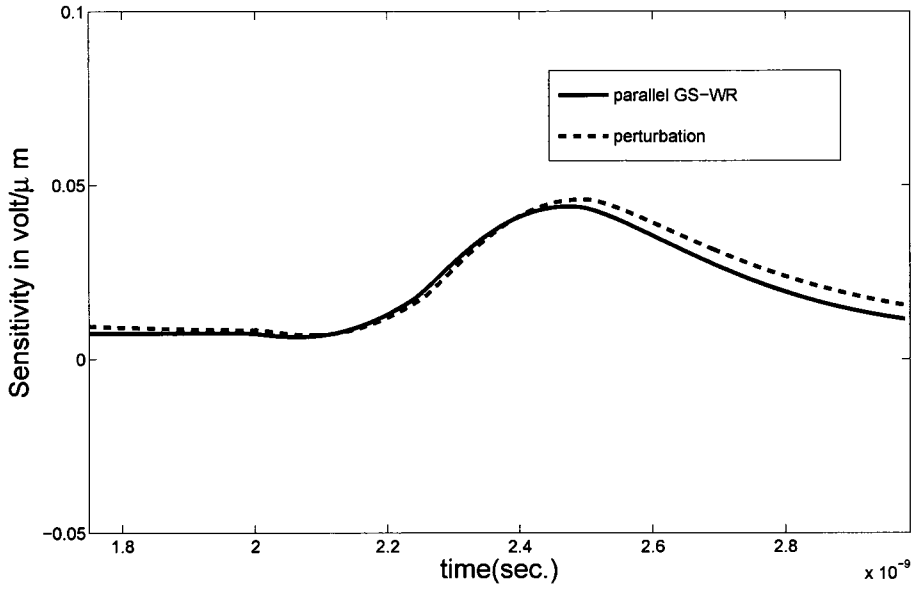

Figure 6.5: Sensitivity Convergence Waveform v/s the Variations in the Width of Conductors for Metal Layer 1 (Example 2)

From the examples presented in this section, we can conclude that the proposed GS-WR based sensitivity algorithm is accurate and converges to the waveforms from the perturbation technique.

Example 3:

In this example, performance of the proposed parallel GS-WR algorithm for sensitivity analysis is demonstrated for a large scale industrial power grid networks, PG_1 and PG_2. Both the power grid networks are based on flip-chip package for microprocessor

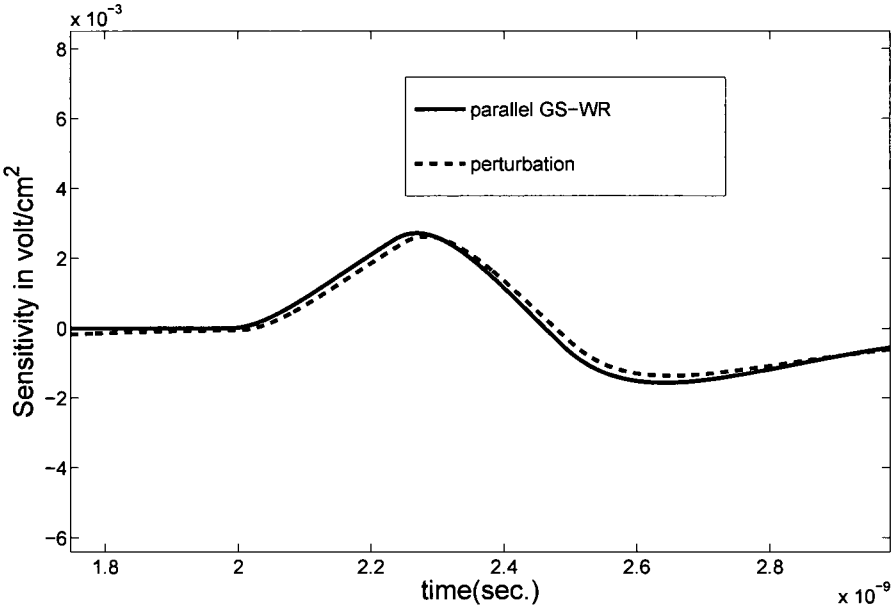

Figure 6.6: Sensitivity Convergence Waveform v/s the Variations in the Overlap Area of the Conductors of Layer 1 and 2 (Example 2)

of dimension $10mm$x$10mm$, built around a 65nm process [27]. The C4 bumps are placed at every 300μm at the top metallization layer. PG_1 is generated from a 4-metal layer PDN resulting in a circuit containing 2.5 million nodes. PG_2 is generated from a 5-metal layer PDN resulting in a circuit containing 7.5 million nodes. The sensitivity parameter chosen for this example is the pitch of metal layer 1 (for details see equations (6.10) to (6.13)). However, the results shown below are applicable for other physical parameters like the width of the metal conductors, and overlap area.

All the simulations for this example were run on the high performance computing

Table 6.4: CPU Cost Comparison for Simulating Circuit $\boldsymbol{PG}_1$ on a Parallel Platform (Example 3)

Circuit $\boldsymbol{PG}_1$: # of nodes= 2.5 millions			
Proposed parallel GS-WR sensitivity method		Perturbation method	Speed-up
Number of CPUs	CPU time (hr:min:sec)	(hr:min)	
1	2:43:22	2:07	**0.78**
2	1:21:28		**1.56**
4	0:40:59		**3.09**
8	0:21:06		**6.02**
16	0:11:29		**11.06**
18	0:10:10		**12.49**

virtual laboratorys (HPCVL) [28] Sun Sparc Enterprise M9000 machines. These machines are high end SMP machines and consist of 64 quad-core 2.52 GHz Sparc64 VII processors. Each of these chips have 4 compute cores, and each core is capable of Chip Multi Threading with 2 hardware threads. These machines have a total of 2TB of memory, which is about 8GB per core, and are meant for very high memory and computation intensive applications. All these machines run Solaris 10 UNIX.

Table 6.5: CPU Cost Comparison for Simulating Circuit $\boldsymbol{PG_2}$ on a Parallel Platform (Example 3)

Circuit $\boldsymbol{PG_2}$: # of nodes= 7.5 millions			
Proposed parallel GS-WR sensitivity method		Perturbation method	Speed-up
Number of CPUs	CPU time (hr:min:sec)	(hr:min)	
1	11:06:21	16:28	**1.48**
2	5:33:12		**2.97**
4	2:47:10		**5.91**
8	1:24:49		**11.65**
16	0:44:39		**22.13**
18	0:40:33		**24.37**

These circuits were simulated for 100 time points using both the proposed parallel GS-WR algorithm, and the perturbation technique. The scalability of both these methods is investigated by running the example on different numbers of processors, and the corresponding CPU costs are given in Table 6.4. The resulting speed-ups are plotted against the number of CPUs, in Fig. 6.7.

The above experiments were also repeated for PG_2. The resulting CPU simulation

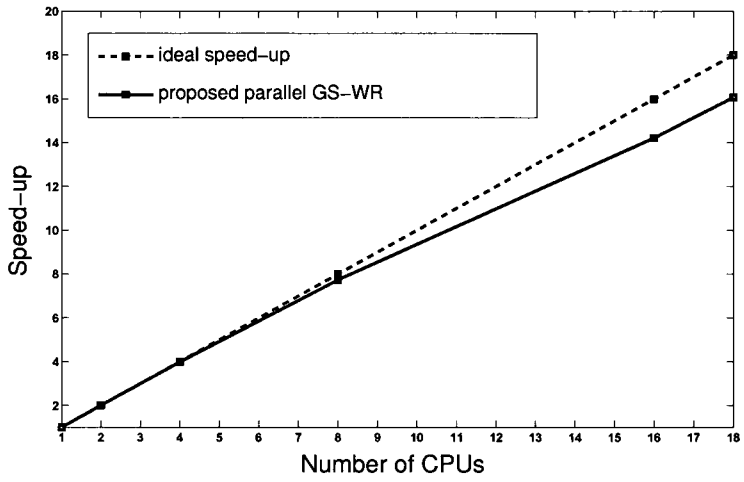

Figure 6.7: Speed-Up Graphs for Circuit PG_1 Using the Proposed Parallel GS-WR Algorithm for Sensitivity Analysis on Single v/s Multiprocessor Environment (Example 3).

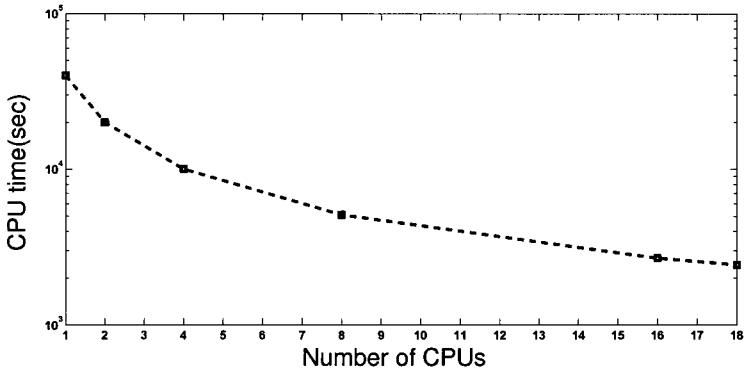

Figure 6.8: Simulation Times for Circuit PG_2 Using the Proposed Parallel GS-WR Algorithm for Sensitivity Analysis with Varying Number of Processors (Example 3).

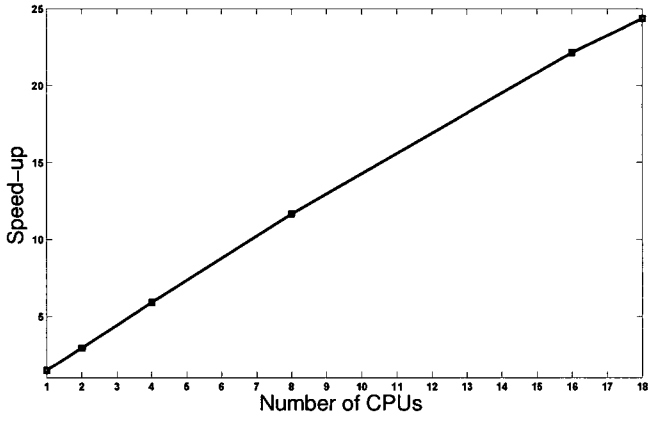

Figure 6.9: Speed-Up Graphs for Circuit PG_2 Using the Proposed Parallel GS-WR Algorithm for Sensitivity Analysis With Respect to Perturbation Technique (Example 3).

times, and the corresponding speed-ups compared to the traditional perturbation method for sensitivity analysis are tabulated in Table 6.5. The CPU times for the proposed parallel algorithm for sensitivity analysis are plotted against the number of CPUs in Fig. 6.8, on a semilog scale. As seen, while using the proposed GS-WR parallel algorithm for sensitivity analysis, the required simulation times reduced considerably with the increasing number of processors. Fig. 6.9 shows the speed-up achieved using the proposed algorithm compared to the traditional perturbation method, with varying number of processors. Fig. 6.10 shows the speed-up achieved with the proposed parallel GS-WR algorithm for sensitivity analysis while using it on a single versus multiprocessor environment. As seen, the algorithm scales very well

with the number of processors.

6.4 Summary

In this chapter, a novel algorithm for sensitivity analysis of power distribution networks has been presented. The proposed method is an extension of the parallel Gauss-Seidel waveform relaxation technique proposed for transient analysis in [1–3]. The BRP scheme and parallel GS-WR have been appropriately applied for sensitivity analysis. The proposed algorithm employs direct sensitivity method for sensitivity analysis.The proposed parallel algorithm exhibits significant speed-ups compared to

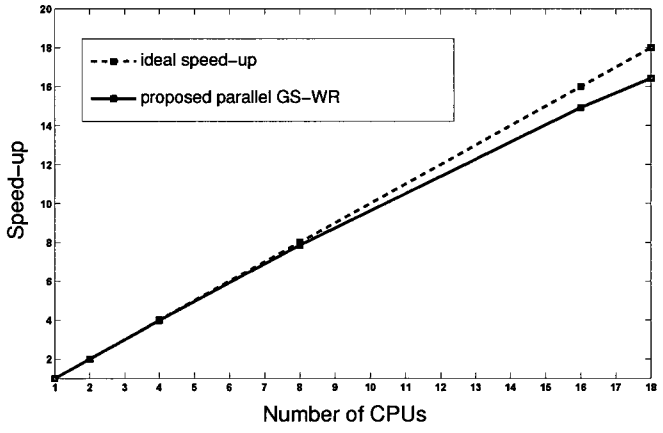

Figure 6.10: Speed-Up Graphs for Circuit PG_2 Using the Proposed Parallel GS-WR Algorithm for Sensitivity Analysis on Single v/s Multiprocessor Environment (Example 3).

conventional perturbation techniques. The speed up of the developed parallel algorithm is independent of the number of sensitivity parameters and depends on the problem size. It provides higher speed up even for a single sensitivity parameter. The proposed parallel algorithm shows good scalability with the increasing number of processors and can be employed for the applications where speed of simulation is desired.

Chapter 7

Conclusions and Future Work

7.1 Summary

The proper operation of integrated circuits (ICs) depends on the design of the power distribution networks supplying power to the on-chip circuitry. For modern ICs, proper analysis of power distribution networks poses a sever challenge to the designers. To address this difficulty, efficient parallel algorithms for the analysis and design of on-chip power grid networks have been developed in this thesis. Specifically, following parallel algorithms based on waveform relaxation (WR) have been developed:

1. **Parallel Transient Analysis of Power Distribution Networks**: A new parallel algorithm, based on waveform relaxation (WR) method, for transient analysis of large scale on-chip power distribution networks (PDNs)

was proposed. Gauss-Seidel WR has been parallelized for power distribution networks. Efficient pipelining schemes have been developed to enhance the performance and CPU utilization of the developed parallel algorithm. Test power grid structures and large scale industrial power grid networks were simulated to demonstrate the speed up and scalability of the developed parallel algorithm.

2. **Parallel Sensitivity Analysis of Power Distribution Networks**: Novel parallel algorithm for sensitivity analysis of large on-chip power distribution networks was developed. The new algorithm is based on waveform relaxation techniques. Novel partitioning schemes, exploiting the localized voltage distribution behavior of the power grids were developed. Parallel Gauss-Seidel WR and enhanced pipelining techniques were also developed for improving the performance of the developed parallel algorithm. Test power grid structures and industrial standard power grid circuits were simulated, to demonstrate the efficiency and scalability of the proposed parallel sensitivity analysis algorithm.

The proposed algorithms are scalable and highly generic in nature. For power grid transient analysis using the proposed WR algorithm, individual sub circuits can be simulated using any direct-solver technique (such as LU or QR decomposition of matrices used with other stable integration method for numerical integration of the

circuit equations). This can also be extended to the proposed sensitivity analysis algorithm for power grid networks.

7.2 Future Work

Through the work in this thesis, many possibilities for future research have become apparent. They are summarized below.

1. **Second Order Effects in Power Grid Networks**: In the proposed algorithm, the power distribution networks has been modeled as a resistive-capacitive network. However, with the increasing operating frequencies of VLSI circuits, the second order effects such as inductive coupling, distributed effects, package-power grid interaction, and electromagnetic interference will become prominent and affect the operation and analysis of power grids. Therefore, it will be critical to include these effects in the early design and simulations of power grid network. Hence, future research in power grid analysis would be to evaluate the influence of these effects on circuit simulation and investigate the viability of waveform relaxation techniques for the same.

2. **Large Scale Parallel Sensitivity Analysis**: One area of future research would be to extend the principles of the proposed waveform relaxation-based

techniques to large-scale sensitivity analysis and use parallel computing to make this analysis computationally efficient.

3. **Alternative Partitioning Schemes**: In this thesis, partitioning scheme based on BRP scheme has been used. However, this partitioning scheme has some limitations as the its subcircuit size is limited by the distribution of the V_{DD} contacts. New and more efficient partitioning schemes can be developed to enhance the performance. Further, dynamic partitioning schemes can also be developed, in which the size of the subcircuits vary in different regions in power grid to adjust for any local or global multirate variations in voltages.

Appendix A

A.1 Parallelization Strategy for Sensitivity Analysis while using Direct Sensitivity Methods

In this appendix, various parallelizing strategies for sensitivity analysis while using direct sensitivity method are examined. The aim of examining each strategy is to find the best possible parallel algorithm which is memory efficient while yielding scalable speed up with an increasing number of processors.

The direct differentiation approach, as described in pseudocode 6.1, can be easily extended to solve for multiple sensitivity parameters in parallel. This is a very simple approach which exhibits inherent coarse-grained parallelism. The idea is to first solve the original circuit equation (6.2) by finding the LU factors of the left-hand side and then using forward-backward (FB) substitutions to obtain the transient circuit solution. Then, a number of sensitivity equations (6.5), corresponding to different

sensitivity parameters, can be solved on different processors in parallel while sharing the same pre-computed LU factors of the original network. This step involves only FB substitutions, since the LU factors are already known. Hence, the parallel computation part of this approach is FB substitutions for various sensitivity calculations, and the serial part is the computation of LU factors of the original network.

In the following, parallelization approaches and issues while using traditional direct sensitivity method is discussed. For the purpose of illustration, an example of coupled MTL structure is considered. The direct method approach similar to the one elaborated in the Section 6.1 has been used. However, the conclusions drawn in this section can be appropriately extended for general cases while using direct sensitivity method.

In the following analysis, lumped segmentation has been used to model fully coupled MTL structures. The MTL structure can be defined by the following MNA equation:

$$\boldsymbol{G}\boldsymbol{X}(t) + \boldsymbol{C}\dot{\boldsymbol{X}}(t) = \boldsymbol{b}\boldsymbol{u}(t), \tag{A.1}$$

where $\boldsymbol{X}(t)$ represents the vector of branch currents and node voltages of the circuit, $\boldsymbol{G}$ and $\boldsymbol{C}$ are the conductance and capacitance matrices for the lumped segmentation MTL structure, $\boldsymbol{u}(t)$ is the vector of inputs to the circuit and $\boldsymbol{b}$ is a selection matrix.

Following the direct sensitivity method, equation (A.1) is differentiated with re-

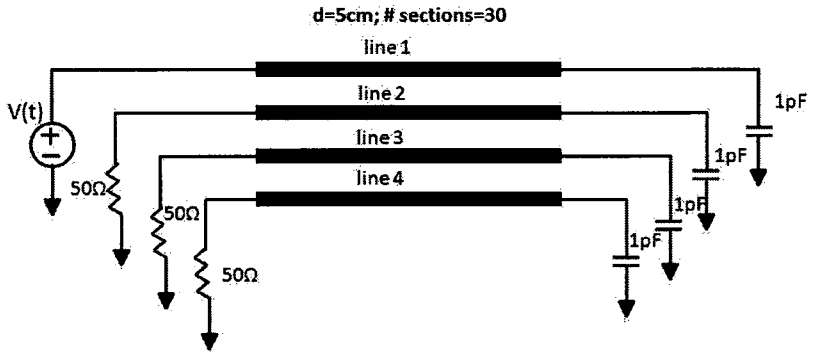

Figure A.1: Circuit With 4-Line MTL (for *Illustrative Example 1*).

spect to the sensitivity parameter α to obtain the sensitivity circuit described by

$$\boldsymbol{G}\boldsymbol{Z}(t) + \boldsymbol{C}\dot{\boldsymbol{Z}}(t) = -\frac{\partial \boldsymbol{G}}{\partial \alpha}\boldsymbol{X}(t) - \frac{\partial \boldsymbol{C}}{\partial \alpha}\dot{\boldsymbol{X}}(t), \tag{A.2}$$

where $\boldsymbol{Z}(t)$ represents the vector of sensitivity of circuit node voltages with respect to the varying parameter α. Next, two examples have been considered to investigate the scalability of the parallel direct sensitivity method, with respect to the number of sensitivity parameters.

Illustrative Example 1:

In this example, 4 fully coupled MTLs are considered. Each line has 30 sections and capacitive terminations as shown in Fig. A.1. The length of the lines is taken to be 5cm. Table A.1 shows the results for this case with 34, 68, 170 and 340 sensitivity parameters. As can be seen from Table A.1 and Fig. A.3, the parallel direct sensitiv-

Table A.1: CPU Time Comparison for Parallel Direct Sensitivity Method for Illustrative Example 1

# of processors	# sens param=34		# of sens param=68		# of sens param=170		# of sens param=340	
	CPU time	Speed up	CPU time	Speed up	CPU time	Speed up	CPU time	Speed up
1	8.52	-	17.15	-	41.43	-	81.25	-
2	4.52	1.88	9.34	1.84	22.67	1.83	44.07	1.84
4	3.22	2.65	5.78	2.96	14.12	2.93	26.22	3.1
8	3.15	2.7	5.77	2.97	12.98	3.19	22.98	3.54

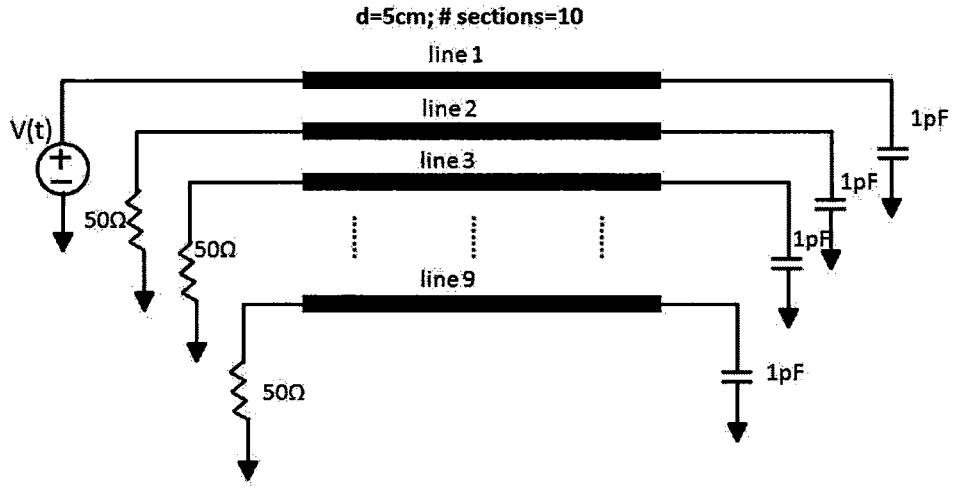

Figure A.2: Circuit With 9-Line MTL (for *Illustrative Example 2*).

ity algorithm exhibits low speed-up and poor scalability with an increasing number of processors. The trend remains the same as the number of sensitivity parameters are increased. As noted, the parallel direct sensitivity method shows dismal speed-up and scalability. The speed-up numbers show a similar trend for larger problem size as illustrated by the next illustrative example.

Illustrative Example 2:

In this example, 9 fully coupled MTLs are considered as shown in Fig. A.2. Each line has 10 sections and capacitive terminations. The length of the lines is taken to be 5cm. Table A.2 shows the results for this case with 34, 68, 170 and 340 sensitivity

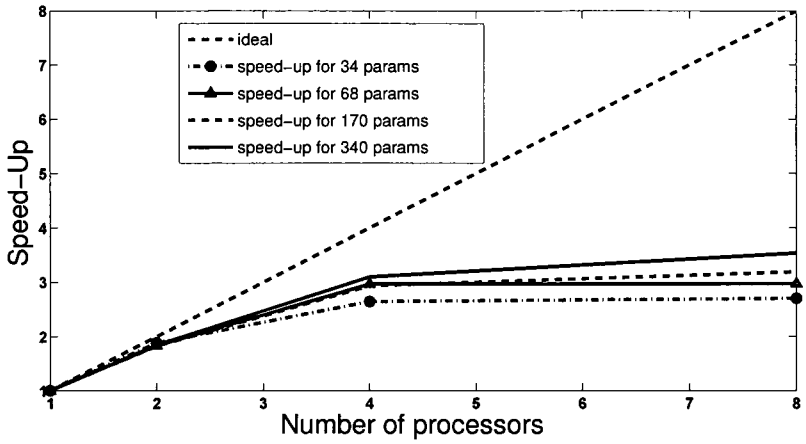

Figure A.3: Speed-Up and Scalability Graph for *Illustrative Example 1.*

---

parameters. Table A.2 and Fig. A.4 show that a larger problem size has little effect on the scalability performance of the parallel direct sensitivity algorithm. The speed up shows only minor improvement for large number of sensitivity parameters.

Table A.2: CPU Time Comparison for Parallel Direct Sensitivity Method for Illustrative Example 2

# of processors	# sens param=34		# of sens param=68		# of sens param=170		# of sens param=340	
	CPU time	Speed up	CPU time	Speed up	CPU time	Speed up	CPU time	Speed up
1	9.64	-	19.2	-	45.67	-	90.45	-
2	4.95	1.95	10.5	1.83	25.54	1.8	48.65	1.86
4	3.66	2.63	6.79	2.86	15.47	2.95	27.57	3.28
8	3.21	3	6.2	3.1	13.89	3.29	21.18	3.97

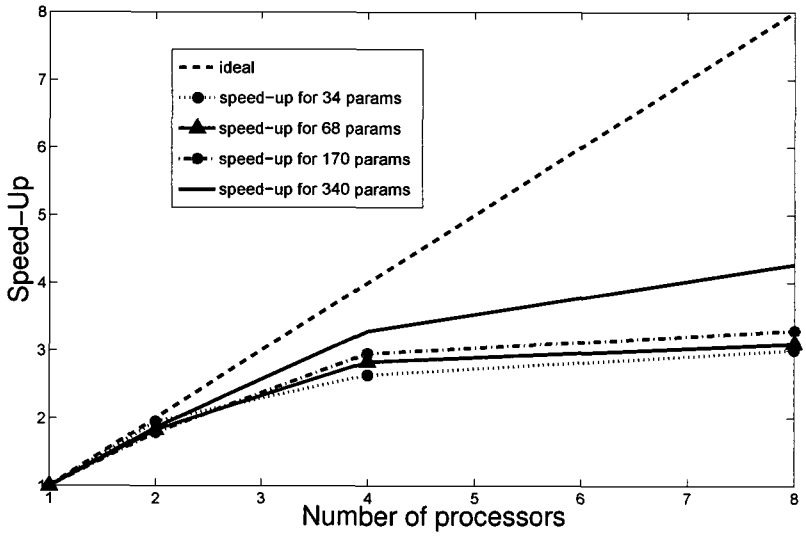

Figure A.4: Speed-Up and Scalability Graph for *Illustrative Example 2.*

References

[1] R. Achar, M. Nakhla, H. Dhindsa, A. Sridhar, D. Paul, and N.Nakhla, "Parallel transient simulation of power grid networks via waveform relaxation," *IEEE Trans. Very Large Scale Integration Syst.*, (publication pending) 2009.

[2] R. Achar, M. Nakhla, A. Sridhar, H. Dhindsa, and D. Paul, "Fast analysis of power distribution networks using waveform relaxation," in *IEEE workshop on Signal Propagation on Interconnects*, May 2009, pp. 1–4.

[3] H. Dhindsa, A. Sridhar, R. Achar, M. Nakhla, and D. Paul, "Transient analysis of power grid networks via waveform relaxation techniques," in *Proc. IEEE MMT-S International workshop series on Signal Integrity and High-Speed Interconnects (IMWS)*, Guadalajara, Mexico, February 2009, pp. 91–94.

[4] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing.* Harlow, UK: Addison-Wesley, 2003.

[5] G. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," in *AFIPS Conference*, New Jersey, USA, April 1967, pp. 483–485.

[6] J. L. Gustafson, "Reevaluating amdahl's law," *ACM*, vol. 31, no. 5, pp. 523–533, May 1988.

[7] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* New York,US: McGraw-Hill, 1992.

[8] A. Mezhiba and E. Friedman, *Power Distribution Networks in High Speed Integrated Circuits.* Massachusetts, US: Kluwer Academic Publishers, 2004.

[9] J. Kozhaya, S. Nassif, and F. Najm, "A multigrid-like technique for power grid analysis," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 10, pp. 1148–1160, October 2002.

[10] M. Zhao, R. Panda, S. Sapatnekar, and D. Blaauw, "Hierarchical analysis of power distribution networks," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 2, pp. 159–168, February 2002.

[11] H. Qian, S.Nassif, and S. Sapatnekar, "Random walks in a supply network," in *Design Automation Conference*, Anaheim, USA, June 2003, pp. 93–98.

[12] H. Qian, S. Nassif, and S. Sapatnekar, "Power grid analysis using random walks," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 8, pp. 1204–1224, August 2005.

[13] C. Zhuo, J. Hu, M. Zhao, and K. Chen, "Power grid analysis and optimization using algebraic multigrid," *IEEE Trans. Computer-Aided Design*, vol. 27, no. 4, pp. 738–751, April 2008.

[14] T. Chen and C. Chen, "Efficient large-scale power grid analysis based on preconditioned krylov-subspace iterative methods," in *Design Automation Conference*, Las Vegas, USA, June 2001, pp. 559–562.

[15] Y. Zhong and M. Wong, "Fast algorithms for IR drop analysis in large power grid," in *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, USA, November 2005, pp. 351–357.

[16] Y. Zhong and M.Wong, "Efficient second-order iterative methods for IR drop analysis in power grid," in *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, November 2007, pp. 768–773.

[17] I. Ferzli, E. Chiprout, and F. Najm, "Verification and co-design of the package and die power delivery system using wavelets," in *IEEE Conference on Electrical Performance of Electronic Packaging*, San Jose, USA, October 2008, pp. 7–10.

[18] K. Sun, Q. Zhou, K. Mohanram, and D. C. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids," in *International Conference on Computer-Aided Design*, San Jose, USA, November 2007, pp. 54–59.

[19] E. Lelarasmee, A. Ruehli, and A. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large scale intergrated circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-I, pp. 131–145, August 1982.

[20] G. Antonini, J. Ekman, and A. E. Ruehli, "Waveform relaxation for the parallel solution of large PEEC model problems," *IEEE Trans. Electromagn. Compat.*, pp. 1–6, July 2007.

[21] G. Gristede, A. E. Ruehli, and C. Zukowski, "Convergence properties of waveform relaxation circuit simulation methods," *IEEE Trans. Circuits Syst.*, vol. 45, no. 7, pp. 726–738, July 1998.

[22] F. Y. Chang, "The generalized method of characteristics for waveform relaxation analysis of lossy coupled tranmission lines," *IEEE Trans. Microwave Theory Tech.*, vol. 37, no. 12, pp. 2028–2038, December 1989.

[23] F. Y.Chang, "Waveform relaxation analysis of RLCG transmission lines," *IEEE Trans. Circuits Syst.*, vol. 37, no. 11, pp. 1394–1415, November 1990.

[24] N. Nakhla, M. Nakhla, R. Achar, and A. Ruehli, "Parallel simulation of high-speed interconnects using delay extraction and transverse partitioning," in *IEEE Conference on Electrical Performance of Electronic Packaging*, Atlanta, USA, October 2007, pp. 237–240.

[25] D.Paul, N. Nakhla, R.Achar, and M.S.Nakhla, "Parallel simulation of massively coupled interconnect networks," *IEEE Trans. Adv. Packag.*, (publication pending) 2009.

[26] T. A. Davis, *Direct methods for sparse linear systems.* Philadelphia, USA: SIAM, September 2006.

[27] Private communication: K. Mihan, Javelin Design Automation, Ottawa, Canada.

[28] High Performance Computing Virtual Laboratory, http://www.hpcvl.org/.

[29] P. Li, "Power grid simulation via efficient sampling-based sensitivity analysis and hierarchical symbolic relaxation," in *Proc. Design Automation Conference*, Anaheim, USA, June 2005, pp. 664 – 669.

[30] C. Zhuo, J. Hu, M. Zhao, and K. Chen, "Power grid analysis and optimization using algebraic multigrid," *IEEE Trans. Computer-Aided Design*, vol. 27, no. 4, pp. 738–751, April 2008.

[31] D. Andersson, L. Svensson, and P. Larsson-Edefors, "Toward a systematic sensitivity analysis of on-chip power grids using factor analysis," in *IEEE workshop on Signal Propagation on Interconnects*, May 2007, pp. 155 – 158.

[32] H. Chen and D. Ling, "Power supply noise analysis methodology for deep-submicron vlsi chip design," in *Proc. Design Automation Conference*, June 1997, pp. 638–643.

[33] T. Pillage, R. Rohrer, and C. Visweswariah, *Electronic Circuit and System Simulation Methods.* New York: McGraw Hill, 1995.

[34] D. Hocevar, P. Yang, T. Trick, and B. Epler, "Transient sensitivity computation for mosfet circuits," *IEEE Trans. Electron Devices*, vol. 32, no. 10, pp. 2165–2176, October 1985.