PARALLEL AND DISTRIBUTED MULTI-ALGORITHM CIRCUIT SIMULATION

A Thesis

by

RUICHENG DAI

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

MASTER OF SCIENCE

August 2012

Major Subject: Computer Engineering

PARALLEL AND DISTRIBUTED MULTI-ALGORITHM CIRCUIT SIMULATION

A Thesis

by

RUICHENG DAI

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

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Peng Li
Committee Members,	Nancy Amato
	Jiang Hu
Head of Department,	Costas N. Georghiades

August 2012

Major Subject: Computer Engineering

ABSTRACT

Parallel and Distributed Multi-Algorithm Circuit Simulation. (August 2012)

Ruicheng Dai, B.S., Zhejiang University

Chair of Advisory Committee: Dr. Peng Li

With the proliferation of parallel computing, parallel computer-aided design (CAD) has received significant research interests. Transient transistor-level circuit simulation plays an important role in digital/analog circuit design and verification. Increased VLSI design complexity has made circuit simulation an ever growing bottleneck, making parallel processing an appealing solution for addressing this challenge. In this thesis, we propose and develop a parallel and distributed multi-algorithm approach to leverage the power of multi-core computer clusters for speeding up transistor-level circuit simulation.

The targeted multi-algorithm approach provides a natural paradigm for exploiting parallelism for circuit simulation. Parallel circuit simulation is facilitated through the exploration of algorithm diversity where multiple simulation algorithms collaboratively work on a single simulation task. To utilize computer clusters comprising of multi-core processors, each algorithm is executed on a separate node with sufficient system resource such as processing power, memory and I/O bandwidth. We propose two communication schemes, namely master-slave and peer-to-peer schemes, to allow for inter-algorithm communication. Compared with the shared-memory based multi-

algorithm implementation, the proposed simulation approach alleviates cache/memory contention as a result of multi-algorithm execution and provides further runtime speedups.

DEDICATION

To my parents

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Peng Li. Dr. Li has supervised, advised and guided me from the very beginning stage of this work, as well as gave me extraordinary experience throughout the research. His dedication to excellence, encouragement to students, and enthusiasm for research, will leave a lasting imprint on me.

I would like to thank other professors as well, who are always willing to discuss with me and give new ideas. Particular thanks to Dr. Amato and Dr. Hu, for their constructive comments on this thesis. Thanks also to my colleagues, department faculty and staff for making my time at Texas A&M University a great experience.

Finally, I am grateful for my family and friends. Thanks to my mother and father for their encouragement and love.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1 Motivation

As a fundamental technology in computer-aided design, circuit simulation provides insights into electronic circuits by leveraging mathematical models to replicate the behavior of an actual electronic device or circuit [1]. In transistor-level time-domain circuit simulation, DC analysis is used to obtain quiescent operating point and transient analysis is employed to compute the time-domain response of the circuit. Accurate, fast and robust transistor-level circuit simulation plays a critical part in the design and verification of digital/analog circuit.

Since 1965, Gordon E. Moore, the co-founder of Intel put forward that the number of transistors on integrated circuits would double every two years. This prophecy, also known as Moore's law, became the guidance of the development of integrated circuit technology for later decades. A typical Very Large Scale Integrated (VLSI) Circuit may integrate millions of transistors and other components in a few square millimeters on a chip. Simulation of large IC designs as well as inherent high accuracy requirements places a heavy burden on circuit simulation. For instance, circuit designers may have to spend several days or even weeks on expensive circuit simulation, which greatly influences the design efficiency.

However, with the recent industry's shift to multi- and many-core processor

---

This thesis follows the style of *IEEE Transactions on Computer-aided Design of Integrated Circuit and System.*

technology, parallel computing is ubiquitous and changing the landscape of computing and data processing. This change has made profound implications on the development of compute-intensive applications. Leveraging the available parallel compute hardware leaves new opportunities and challenges to large-scale circuit simulation.

1.2 Previous Work and Limitations

Parallel circuit simulation is not a new topic. The two key challenges of applying parallelism to CAD area are parallel algorithm development and parallel program implementation. Prior work attempted to realize more parallelism from several different perspectives.

Parallel device evaluation and matrix solve [2][3] are the most direct methods. Device evaluation and matrix solve are the most time consuming parts in simulation and dominate the total simulation time. It is straightforward to leverage more threads/CPUs in these two parts to gain large parallelism. However, the speedup is not linear due to the characteristic of the circuit and multi-core computers. Creating threads, terminating and synchronization also will add some overhead to the system.

There also have been attempts to realize parallel capabilities in a single simulation algorithm. Waveform pipelining approach [4] simultaneously computes circuit solutions at multiple adjacent time points in a way resembling hardware pipelining. Circuit decomposition can divide a large circuit into several small sub-circuits which can be solved in parallel. However, decomposition-based circuit simulation algorithms like multilevel newton algorithm [5] and waveform relaxation

algorithm [6] have issues in terms of convergence. In addition, these two methods exploit fine-grained parallelism, hence require large programming effort.

The multi-algorithm parallel approach [7] exploits inter-algorithm parallelism by running several simulation algorithms on a shared-memory multi-core machine simultaneously.

However, most of these works are carried on multi-core shared memory machines. While the methods are gaining the benefits from these platforms, like low on-chip communication overhead, they also have to pay a price for the drawbacks. For instance, the memory on a multi-core machine is shared by all processes/threads and the number of CPUs on one computer is limited due to the manufacture process and power consumption. Hence, memory contention is inevitable as well as severe thread contention when the number of threads is greater than the number of CPUs. The system performance will suffer noticeable degradation.

Computer clusters offer a promising computing solution to address ever complex, computationally intensive simulation problems with sufficient computing resources and high memory bandwidth.

1.3 Overview and Organization

In this thesis, we propose a distributed and parallel multi-algorithm circuit simulation where multiple simulation algorithms are mapped on separated nodes in a supercomputer and work on the same simulation task with effective communication schemes to realize the on-the-fly synchronization and exploration of algorithm diversity. With sufficient

computing resource utilized for parallel device evaluations and parallel matrix solvers in each algorithm, simulation runtime is further reduced. As a coarse-grained parallel approach, the proposed distributed circuit simulation requires less programming effort and is applicable for an increasing number of simulation algorithms.

This thesis is organized as follows. In Chapter 2, we introduce the background for time-domain circuit simulation and parallel computing. Then the principle of multi-algorithm circuit simulation as well as the diversity of numerical integration methods and nonlinear iterative methods will be discussed in Chapter 3. In Chapter 4, we will present the details of the MPI based parallel and distributed circuit simulation. In Chapter 5, the platform where the experiments are carried on and experimental results will be given. Finally, conclusions are drawn in Chapter 6.

2. BACKGROUND

2.1 Transistor-level Circuit Simulation

Transistor-level time-domain circuit simulation, a computer-aided design tool, greatly improves design efficiency and reduces the labor intensity in digital/Asic VLSI circuit design. Figure 1 is a flow chart of digital/Asic circuit design.

First, system specifications and requirements need to be completed. A graph editor or text editor is used to describe the circuit's structure and behavior. After the behavioral description, synthesis realizes the automatic conversion from high level abstraction to low level description where RTL code is translated to a gate-level circuit. Physical design including floorplanning, placement and routing is then carried out to generate the layout of the design. At last, manufacturing process fabricates designs onto silicon dies which are packaged into ICs [1].

Transistor-level circuit simulation can be performed at the circuit design level based on pre-layout schematic. Also, it may be performed after the post-layout circuit netlists are extracted out. It is not surprising that simulation plays a vital part in predicting circuit performance and rejecting a failing design due to transistor-level circuit simulation also plays an important role in the design of analog and RF circuits.

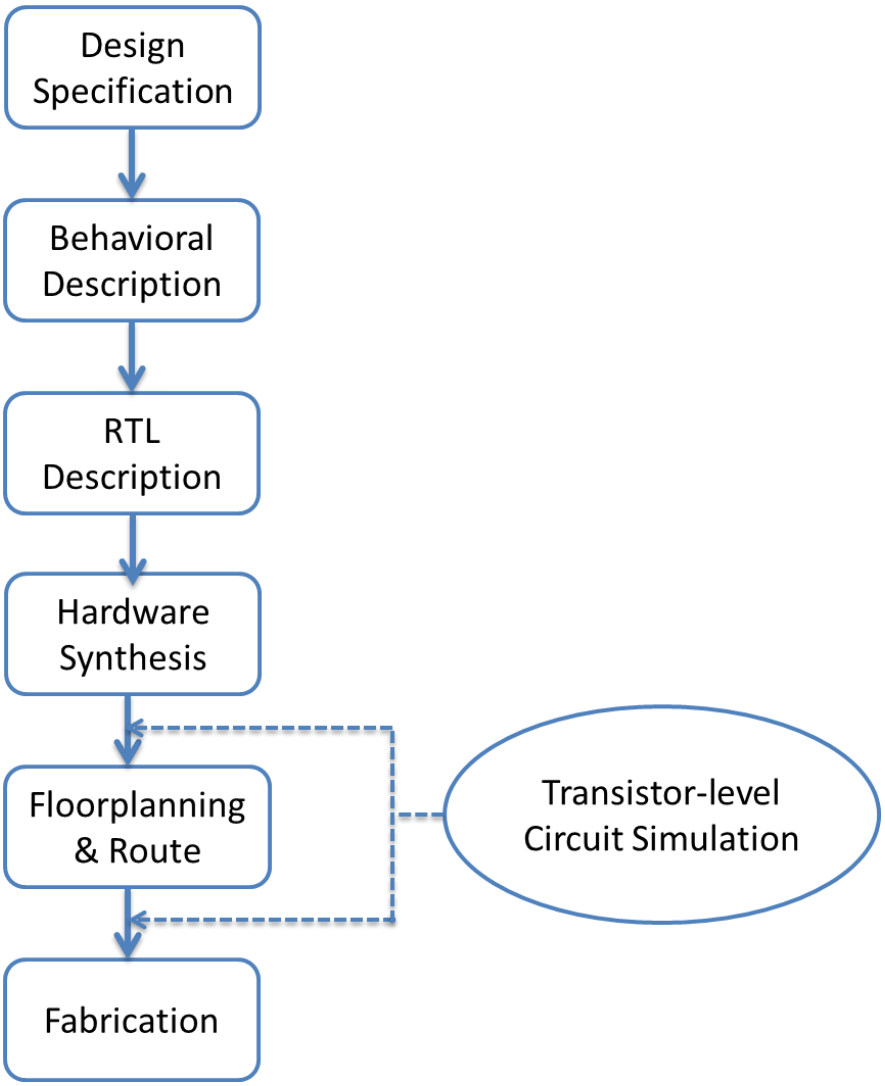

Figure 1. Transistor-level **c**ircuit simulation in digital/Asic design flow.

In transistor-level circuit simulation, circuit analysis problem is formulated according to circuit structure, device parameters and analysis requirements. KVL (Kirchhoff's voltage laws) and KCL (Kirchhoff's current laws) are two basic principles in simulation. Hence, an electronic circuit can be described as a differential-algebraic equation,

$$\frac{d}{dt}q(x)+f(x)=u(t) \tag{2.1}$$

here, $u(t)$ is the input vector, $x(t)$ is the vector of nodal voltages and branch currents. $q(x)$ and $f(x)$ corresponding to dynamic elements and static elements are nonlinear functions. Regarding equation (2.1), the existence of nonlinear functions, $q(x)$ and $f(x)$ is due to the fact that the transistors in the CMOS technology are nonlinear elements with complex nonlinear characteristic. The differential operation represents the behavior of energy storage components like capacitors and inductors which have delay in following the changes of input sources.

For instance, a simple circuit in Figure 2 can be described as equation (2.2)

$$\begin{bmatrix} \frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_4} & -\frac{1}{R_2} \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_3} \end{bmatrix} * \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \frac{E}{R_4} \\ 0 \end{bmatrix} \tag{2.2}$$

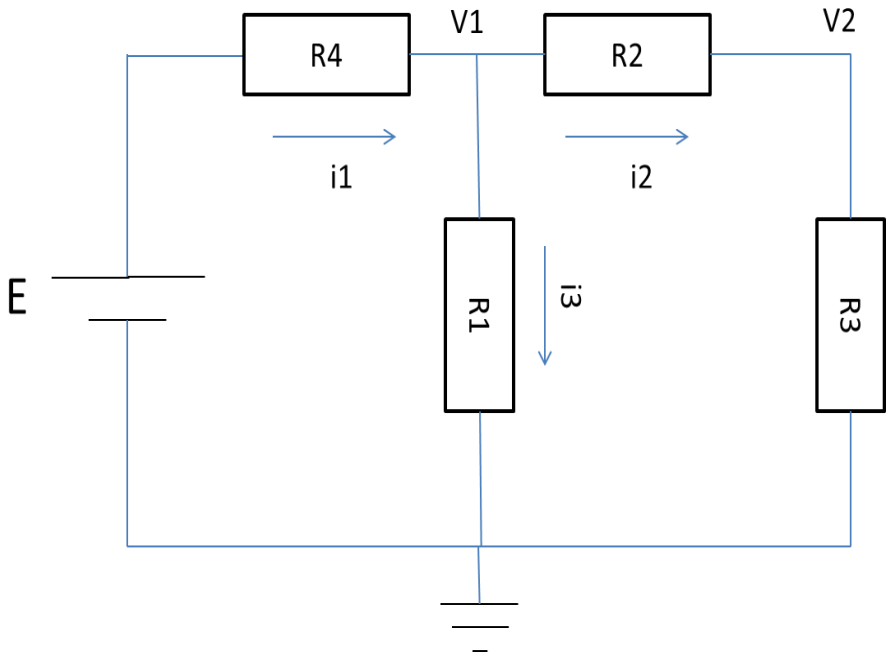

Figure 2. A simple circuit.

To solve equation (2.1), DC analysis is used to obtain an initial operating point. In DC analysis, all the dynamic circuit elements are removed and a nonlinear iterative method is applied to get the solution converged in several iterations. Then a numerical integration method is applied to calculate the transient solutions. At each time point, transient analysis, similarly, needs to utilize the nonlinear iterative method to obtain a converged solution. In other words, by adopting a numerical integration formula, the time-domain transient response of the circuit is obtained by solving a sequence of equivalent nonlinear DC problems sequentially at all time points [8]. The flow chart of the circuit simulation is shown in Figure 3.

In transistor-level circuit simulation, device evaluation and matrix solve are the two most time consuming parts. At each iteration in a single time point, device

evaluation is performed to obtain equivalent mathematical models of circuit components. The evaluation requires numerous computations, especially for nonlinear components such as diodes, transistors, nonlinear resistances and nonlinear capacitances which have a large amount of device model derivatives. For instance, a diode's voltage and current can be represented as

$$I_D = I_S(e^{\frac{V_D}{V_T}} - 1) \tag{2.3}$$

Here, I_S is the reverse bias saturation current and V_T is the thermal voltage. The model of the device has an important position in the whole procedure of circuit analysis because the accuracy of simulation results depends on the precision of the model significantly.

Matrix solve is then applied to obtain the solution for that specific iteration. We LU decompose the matrix to solve the equations. When the coefficient matrix is a sparse matrix, the time complexity of solving the equations will be approximately $O(n)$ [9], here n is the number of the nodes in the circuit.

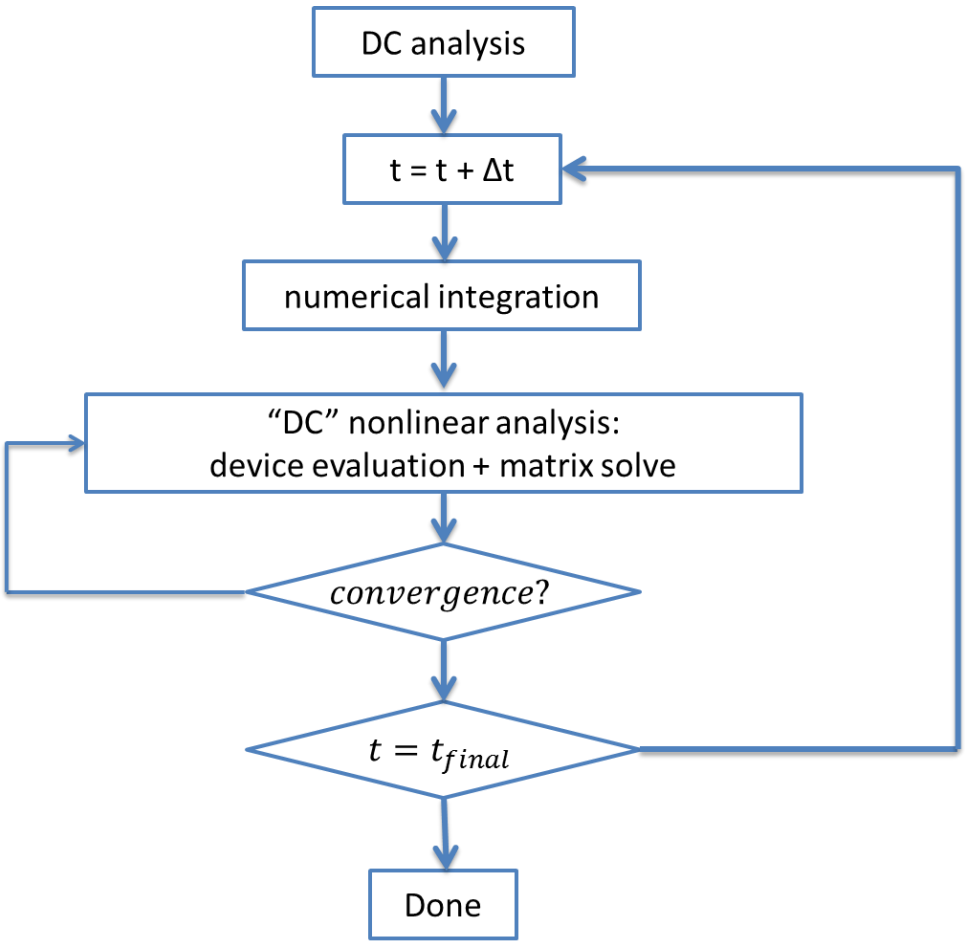

Figure 3. Work flow of the transient circuit simulation.

2.2 Parallel Computing

From the perspective of computer architecture, symmetric multiprocessor (SMP) machine is a system with two or more homogeneous processors on one chip, sharing memory subsystem and bus structure. Although multiple CPUs are running at the same time, they perform as a single machine. The system distributes the tasks in a queue symmetrically over multiple CPUs, thus greatly improving data processing ability of the whole system. Computer clusters emerged as a result of developments of low cost microprocessors and high speed networks. Many independent computer nodes are connected to each other in the cluster through fast local area networks. One computer node can be a single processor or a multiple-processor system, which has memory, I/O devices and operating system. The system can provide a fast and reliable service solution, which can hardly be obtained even through a very expensive shared memory system.

For these parallel platforms, Pthreads and MPI are two most popular parallel programming APIs. POSIX threads [10], commonly known as Pthreads, specifies a set of interfaces (functions, header files) for threaded programming where a single process can create multiple threads. Every thread can be assigned different kind of work and run independently. These threads share data and heap segments, but each thread has its own stack to store automatic variables.

MPI, a kind of Message Passing Interface released in May 1994, is actually a standard of message passing function library [11]. It absorbs benefits from many existing message passing function libraries and becomes one of the most popular parallel

programming environments, especially for distributed storage computers and network-based workstations. MPI has many advantages in providing the necessary conditions for the development of parallel software industry:

- portable and flexible
- complete asynchronous communication function.
- formal, detailed and precise definition

In the MPI based programming model, a fixed set of processes are created in the initialization of the program. Processes receive and send massages by calling library functions. These processes can execute the same or different code paths, correspondingly called single program multiple data (SPMD) or multiple program multiple data (MPMD). Communications between the processes can be point-to-point or collective.

3. MULTI-ALGORITHM PARALLELISM

3.1 Multi-Algorithm Parallelism

From the foregoing discussion, the transient circuit simulation problem can be formulated as equation (3.1).

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

In a circuit simulation algorithm, one nonlinear iterative method is utilized to linearize the nonlinear functions and one numerical integration method replaces differential operation with difference operation. Newton Raphson and Successive Chord are typical nonlinear iterative methods while Backward Euler, Gear2 and DASSL are classic numerical integration methods. A variety of simulation algorithms are then generated within a set of combination between these two kinds of methods. SPICE (Simulation Program with Integrated Circuit Emphasis) [12] is taking Newton-Raphson and Backward Euler as its basic circuit simulation algorithm. It is a general-purpose, open source electronic circuit simulator for integrated circuit and board-level design. Compared to Newton-Raphson and Backward Euler algorithm, Successive Chord is a higher speed simulation algorithm. While the algorithm pool provides a great diversity, it also brings in the complexity in choosing an optimal algorithm for a specific circuit because the algorithms behave quite differently for different kinds of circuits, even in different stages on the same circuit during the whole simulation time.

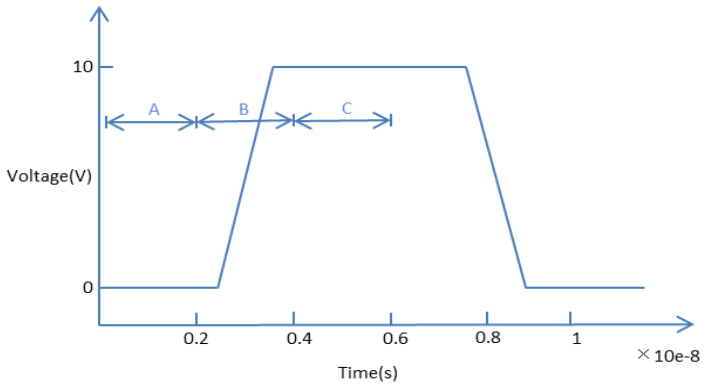

Figure 4. A sample for circuit simulation result.

Figure 4 is simulation results obtained by using SC algorithm and Newton + BE algorithm for inverter chain circuit. During the simulation, we find SC algorithm prints out results much faster on part A and C but slower on part B. From the figure above, we can see the waveform remains stable during parts A and C. Considering SC algorithm's advantage, it can converge very quickly and the cost for each iteration is very small by using a constant Jacobian matrix. In part B, the waveform changes significantly, SC algorithm needs a large number of iterations to converge to the final solution at every time step. Although the cost for each iteration is still small, the time spent on one time step is increasing significantly. When the waveform gets steeper, SC probably will diverge. Inspired by this observation, we know an optimal solution will be obtained if the benefit of SC algorithm on parts A and C is exploited as well as the benefit of Newton + BE algorithm on part B.

Consequently, we refer to the multi-algorithm approach in [7] and propose a new approach that builds on a distributed memory platform to run multiple simulation

algorithms on multiple computer nodes in parallel to exploit the diversity of these algorithms.

To illustrate, we assume two algorithms are initiated on the same circuit simulation. In Figure 5, part A is corresponding to the first time period while part B is the second period. In the first period, algorithm SC is the fastest due to the reason discussed, it can inform its results to algorithm BE + Newton at the end of the first period. With this faster solution, Algorithms BE + Newton can skip its slow part and begin its next period calculation. In part B, Algorithms BE + Newton turns out to be faster and it shares the solution with algorithm SC. In this way, when we adopt more algorithms, we are picking out the best performing algorithm for every small period along the whole simulation and all algorithms' benefits are explored and simulation speed will be optimal.

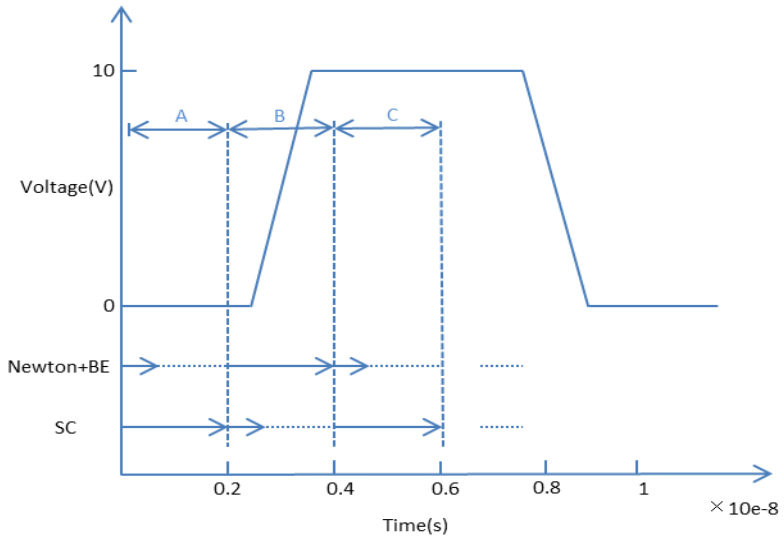

Figure 5. Illustration of the multi-algorithm parallelism

Concerning the communication granularity, if we set the interval as whole simulation time, the system will perform as picking out the fastest simulation algorithm for the simulation task. The diversity will not be fully exploited. However, if we choose a small interval, the communication will be frequent and influence the calculation speed as mutual memory access conflicts are increasing. Hence, there exist tradeoffs between efficiency and communication frequency. In the implementation, we need to choose a reasonable granularity and make the information sharing among all the algorithms efficient. This will be discussed in Chapter 4.

3.2 Simulation Algorithms

In this section, we discuss the advantages and disadvantages of different nonlinear iterative methods and numerical integration methods as well as their roles in simulation algorithm selection.

3.2.1 Diversity in Nonlinear Iterative Methods

At a single time point, the equation (3.1) can be represented as equation (3.2).

$$F(x) = 0 \tag{3.2}$$

A. Newton-Raphson

Newton-Raphson is an effective method in solving nonlinear equations [12]. The solution at $k+1$ iteration is determined by equation (3.3).

$$J(x_k)(x_{k+1} - x_k) = -F(x_k) \tag{3.3}$$

here, $J(x_k)$ is called the Jacobian matrix.

$$J(x_k) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & & \frac{\partial F_2}{\partial x_n} \\ \cdot & & & \\ \cdot & & & \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix} \tag{3.4}$$

Assuming k th iteration's solution is known, the Jacobian matrix and $F(x_k)$ can be calculated by device evaluation, then $(k+1)$th solution is extracted by solving the equation (3.3). If the difference between solutions at iteration $k+1$ and k is smaller than a given threshold, it is accepted as the converged solution. If not, we need to proceed to the next iteration. For instance, $r1$ is the root of equation $f(x)=0$ in Figure 6. The initial solution is assumed at point $P_0(x_0, y_0)$, x_1 is obtained by using the tangent line 1 which is corresponding to equation (3.3). However, y_1 is larger than expected. The next solution x_2 is calculated based on point P_1 similarly.

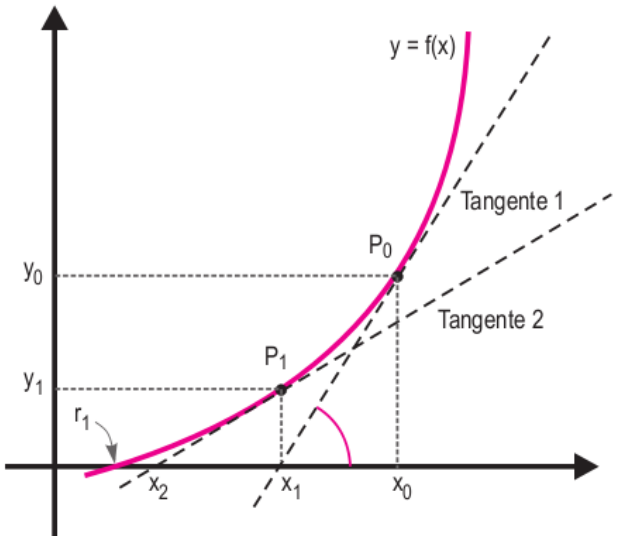

Figure 6. Newton-Raphson method.

When x_k is close to the exact solution, it can be proved that [12]

$$\delta x_{k+1} = C(\delta x_k)^2 \tag{3.5}$$

Here C is constant. Hence, Newton's method has a quadratic convergence rate.

When Newton's method is applied in circuit simulation, its Jacobian matrix needs to be recalculated by evaluating all the devices and decomposed in each iteration. There are a large number of expensive derivative computations. Although Newton method is robust with the quadratic convergence rate, the cost for each iteration is really high and the simulation time at one step is large.

B. Successive Chord method

Another nonlinear iterative method is Successive Chord method (SC) [13]. It can be represented as

$$J_{sc}(x_{k+1} - x_k) = -F(x_k) \tag{3.6}$$

here, the Jacobian matrix J_{sc} is constant. In the following Figure 7, we can get x_1 by using the tangent line 1 which is corresponding to equation (3.6). The final solution x_2 will be obtained in next iteration based on point P_1 . The obvious difference is that the tangent lines are parallel.

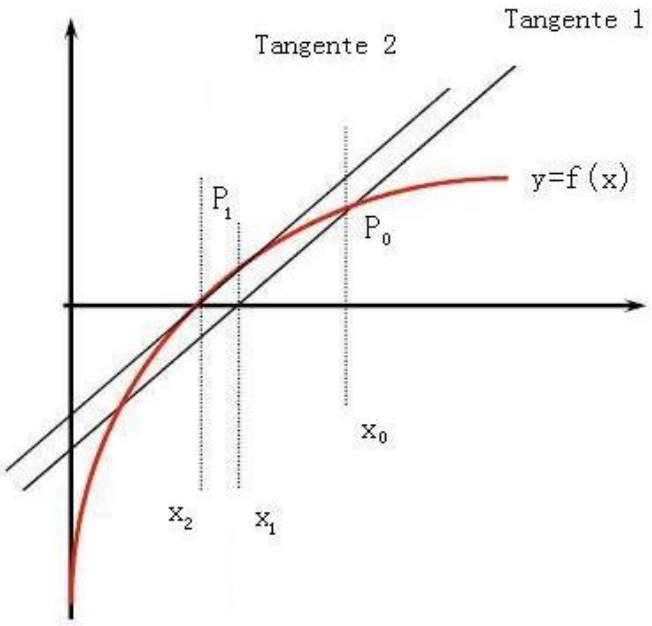

Figure 7. Successive Chord method

Compared to Newton Raphson, SC method's advantage is that it uses constant Jacobian matrix J_{sc} in simulation. The Jacobian matrix is constructed, decomposed at the beginning and the lower upper triangular (LU) factors are stored to reuse efficiently. So the method does not need to calculate the derivative of device equations during the whole simulation. Consequently, the cost for each iteration in SC method is very small.

However, the convergence rate of the SC method is linear which means for every

time step, the method probably needs more iterations. The strict convergence criteria for SC method is

$$\left\| I - J_{sc}^{-1} J_F(v^*) \right\| \le 1 \tag{3.7}$$

Here, I is identity matrix, J_{sc} is chord value, $J_F(v^*)$ is the exact Jacobian matrix.

Consequently, the J_{sc} matrix should be selected wisely. Otherwise this method will probably diverge. According to our research, SC method is hard to converge for analog circuits which have greater changes compared to the combination circuits.

3.2.2 Diversity in Numerical Methods

In transient analysis, equation (3.1) may be represented as a first order differential equation:

$$\dot{x} = f(x,t) \qquad t_0 \le t \le T \tag{3.8}$$

with initial condition:

$$x(t_0) = x_0$$

Here, $\dot{x}$ is the derivative of x, t is the time variable. The initial solution $x(t_0) = x_0$ is solved by DC analysis. In order to solve the differential-algebraic equations, first we need to discretize $[t_0, T]$ to several distinct time points $(t_0, t_1, t_2, \cdots t_n \cdots T)$. Then we use the difference equation to replace the differential equation to get the approximate values at these points $(x_0, x_{1,2}, \cdots x_n \cdots x_m)$. For the solution at t_{n+1}, the number of the previous

solutions $(x_n, x_{n-1}, \cdots)$ used is determined by the numerical methods which can be classified into one-step and multi-step methods.

A. One-step method

Backward Euler is a one step method [12] with

$$x_{n+1} = x_n + h_n x'_{n+1} \tag{3.9}$$

The local truncation errors (LTEs) is

$$LTE_{BE} = -h_n^2 \frac{x''(\xi)}{2} \tag{3.10}$$

here, $h_n = t_{n+1} - t_n$. In circuit simulation, a fixed step-size method is adopted if h_n is fixed as a reasonable value. There also exists variable step-size method for Backward Euler. After an acceptable value is decided as the bound for local truncation error ε, variable h_n is calculated as

$$h_n = \sqrt{\frac{2\varepsilon}{x''(\xi)}} \tag{3.11}$$

Here, $x''(\xi)$ is second order derivative. x_{n+1} is calculated by equation (3.9). If the local truncation error at t_{n+1} is smaller than ε, the solution is acceptable. Otherwise, it will be abandoned and the solution needs re-computation with a smaller h_n until the solution satisfies the error tolerance ε. The variable step-size method enhances Backward Euler method with a larger time step.

Forward Euler is also a one step method with

$$x_{n+1} = x_n + h_n x'_n \tag{3.12}$$

It does not include x'_{n+1} so the calculation is explicit and simple. The solution at any time can be obtained only by its previous solutions, which contributes to its fast speed as well as low robustness.

Another one step method is Trapezoidal [14]. The formula is

$$x_{n+1} = x_n + \frac{h_n}{2}(x'_n + x'_{n+1}) \tag{3.13}$$

with local truncation errors (LTEs) as

$$LTE_{TR} = -h_n^3 \frac{x'''(\xi)}{12} \tag{3.14}$$

It has smaller local truncation error and larger step size.

B. Multi-step methods

Muliti-step methods employ the solution $(x_n, x_{n-1}, \cdots x_{n-p+1})$ at points $(t_n, t_{n-1}, \cdots t_{n-p+1})$ in numerical integration:

$$x_{n+1} = \sum_{i=1}^{p} \alpha_i x_{n-i+1} + \sum_{i=0}^{p} \beta_i x'_{n-i+1} \tag{3.15}$$

p is the order of the integration method.

Gear2 [15] method uses the following formula to get the solution at t_{n+1}.

$$x_{n+1} = -x_{n-1} \frac{h_{n+1}^2}{h_n(2h_{n+1} + h_n)} + x_n \frac{(h_{n+1} + h_n)^2}{h_n(2h_{n+1} + h_n)} + x'_{n+1} \frac{h_{n+1}(h_{n+1} + h_n)}{2h_{n+1} + h_n} \tag{3.16}$$

Here, $h_{n+1} = t_{n+1} - t_n, h_n = t_n - t_{n-1}$, the local truncation error is

$$LTE_{Gear2} = -\frac{h_{n+1}^2 (h_{n+1} + h_n)^2}{6(2h_{n+1} + h_n)} x'''(\xi) \tag{3.17}$$

Here $t_n \le \xi \le t_{n+1}$. Compared to Backward Euler, Gear2 has more complicated integration formula and is much faster with smaller LTE and larger time step size.

DASSL [16], a variable-order variable-stepsize method, uses the predictor and corrector to solve the differential equation. The predictor for a k th order formula is generated by interpolating the last $k+1$ solutions.

$$\omega_{n+1}^P(t_{n-i}) = x_{n-i} \qquad i = 0,1,...,k. \tag{3.18}$$

Hence, the solution at time $n+1$ can be predicted by using the predictor function ω_{n+1}^P,

$$x_{n+1}^{(0)} = \omega_{n+1}^P(t_{n+1}) \qquad x'^{(0)}_{n+1} = \omega'^P_{n+1}(t_{n+1}) \tag{3.19}$$

The corrector polynomial ω_{n+1}^C is an interpolation of the predictor at last k time points and can be solved by the equation (3.20) ,

$$\alpha_s(\omega_{n+1}^C - x_{n+1}^{(0)}) + h_{n+1}(\omega'^C_{n+1} - x'^{(0)}_{n+1}) = 0 \tag{3.20}$$

here $\alpha_s = \sum_{j=1}^{k} \frac{1}{j}$, h_{n+1} is predicted step size for t_{n+1}.

After the corrector ω_{n+1}^C at t_{n+1} is obtained, the circuit solution is solved by equation (3.21) with LTE applied to determine x_n is accepted or not.

$$F(t_{n+1}, \omega_{n+1}^C(t_{n+1}), \omega'^C_{n+1}(t_{n+1})) = 0 \tag{3.21}$$

DASSL uses the LTE to control the step size and the integration order dynamically. Before calculating x_n, DASSL utilize the existing step size and the order

k to estimates the LTE at t_n. With the estimated LTE, DASSL determines the order k' for the next time step. After x_n is solved with above equations, k' is used to solve the next time point solution or recompute x_n based on whether x_n is accepted or not. DASSL has very complex control scheme to maintain stability and is possible to achieve significant speedup.

3.2.3 Algorithm Selection

About the nonlinear iterative methods, we will use the Newton-Raphson and Successive Chord method.

In the numerical methods, the values we got at $(t_0, t_1, t_2, \cdots t_n \cdots T)$ is approximation to the exact values, they are actually $(x_0^*, x_1^*, x_2^*, \cdots x_n^* \cdots x_m^*)$. The errors are introduced by two ways. First, local truncation error is brought in because at time t_{n+1}, we abandon the high order differential item. Second, we get the solution at time t_{n+1} with the previous solutions $(x_n, x_{n-1}, \cdots)$ which we assume are exact values. However, these solutions are approximations because of the LTE. Hence, the errors may accumulate. If the influences of the previous errors on later time pointes do not increase with time, this method is stable. If the errors are accumulated and exceed the error limit, the method is not stable.

In order to clarify this, we introduce a test equation,

$$x' = \lambda x \tag{3.22}$$

If we apply the Forward Euler to the test equation, we will get

$$x_{n+1} = x_n + \lambda x_n h_n = x_n(1 + \lambda h_n) = x_0(1 + \lambda h)^n \tag{3.23}$$

When error at the initial solution is assumed as ε_0, the error at time t_{n+1} is

$$\varepsilon_{n+1} = \varepsilon_0(1 + \lambda h)^n \tag{3.24}$$

here $\lambda < 0$ and real. Consequently, when $|1 + h\lambda| \le 1$ or $0 < h < 2|\lambda|$, ε_{n+1} is bounded and the method is stable. If we represent $|1 + h\lambda| \le 1$ in the complex plane of $h\lambda$, it will be like Figure 8(a). The shaded part is called stability region.

A stability concept, called Absolute Stability, specifies that a method is absolutely stable if the region of the absolute stability covers the entire left plane as in Firgure 9. According to this concept, Forward Euler is unstable while Backward Euler, Trapezoidal method and fixed step size Gear2 method in Figure 8(b)(c)(d) are unconditionally stable.

Actually, stability and local truncation error are two major considerations in selecting numerical integration methods. BE is robust and easy to implement, with large local truncation error and small time step size. Fixed step size Gear2 has much smaller local truncation error and larger time step size. However, Gear2 is much more complex to implement and brings in a large computation cost at every time point. The stability of the DASSL method is more difficult to analyze. According to the experiments, DASSL is stable in most cases as Figure 9 and potentially leads to the largest time step size.

In practice, the performance index of a particular algorithm is determined by the circuit type and input signal. It is difficult to tell which one is the optimal before executing it one time. In the system, we choose Newton-Raphson method (Newton) as a

solid base for the system and Successive Chord method (SC), Gear2 + Newton and DASSL + Newton as aggressive algorithms to speed up the whole system.

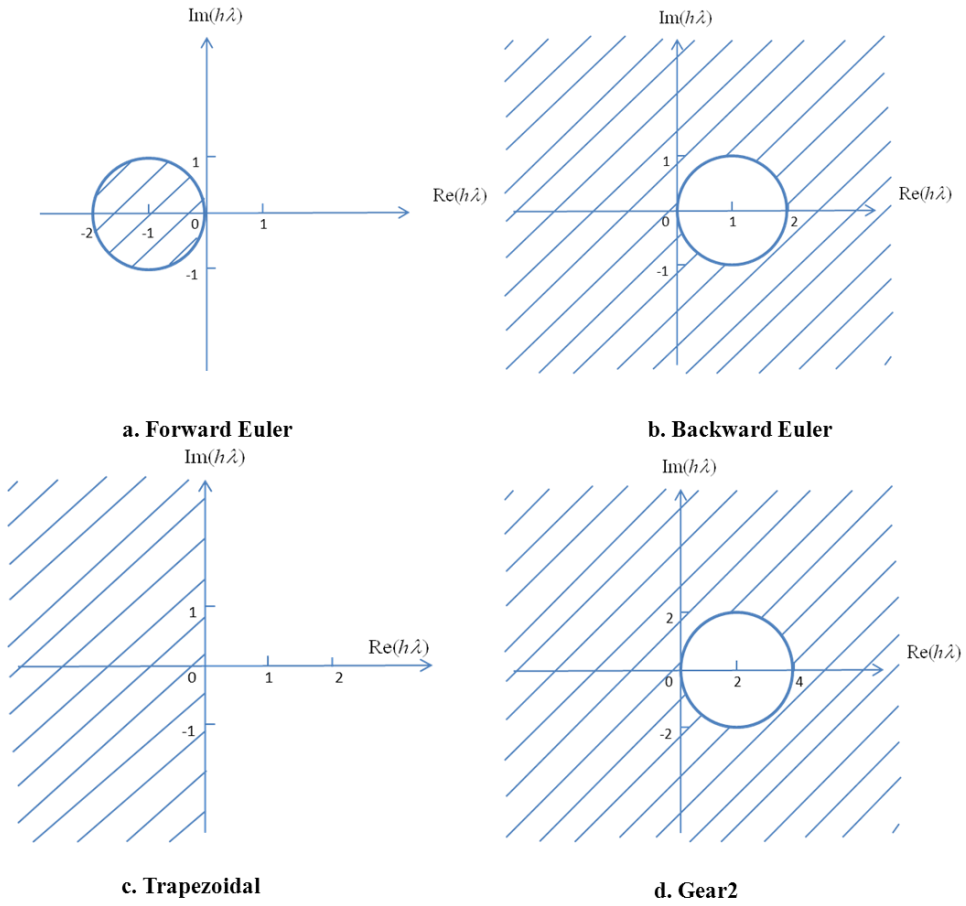

Figure 8. Stability region of numerical integration methods.

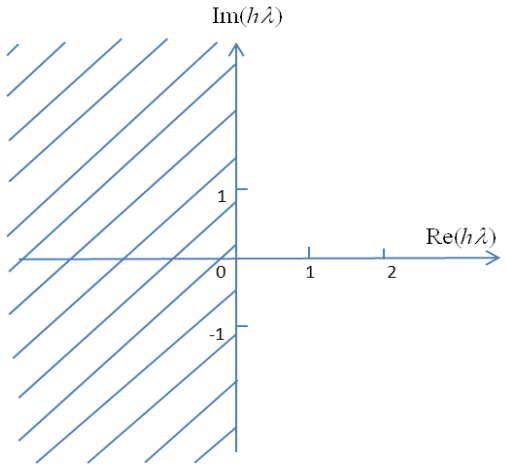

Figure 9. Stability region of Absolute Stability.

4. HIERARCHY OF PARALLEL AND DISTRIBUTED CIRCUIT SIMULATION

The hierarchy of parallel and distributed circuit simulation, built on a computer cluster in Figure 10, adopts two levels of parallelism, inter-algorithm parallelism and intra-algorithm parallelism. At the higher level of parallelism, multiple simulation algorithms are performed in parallel on separate computer nodes with MPI methods transferring data between them to exploit the algorithm diversity. The cloud in Figure 10 represents the communication structures between nodes. Two MPI communication structures are proposed, namely master-slave structure and peer-to-peer structure, with different characteristic corresponding to the type and size of circuit. At the lower level of parallelism, each algorithm has full control of all resources like CPUs, memory bandwidth and I/O, which allows it to reach to high intra-algorithm parallelism.

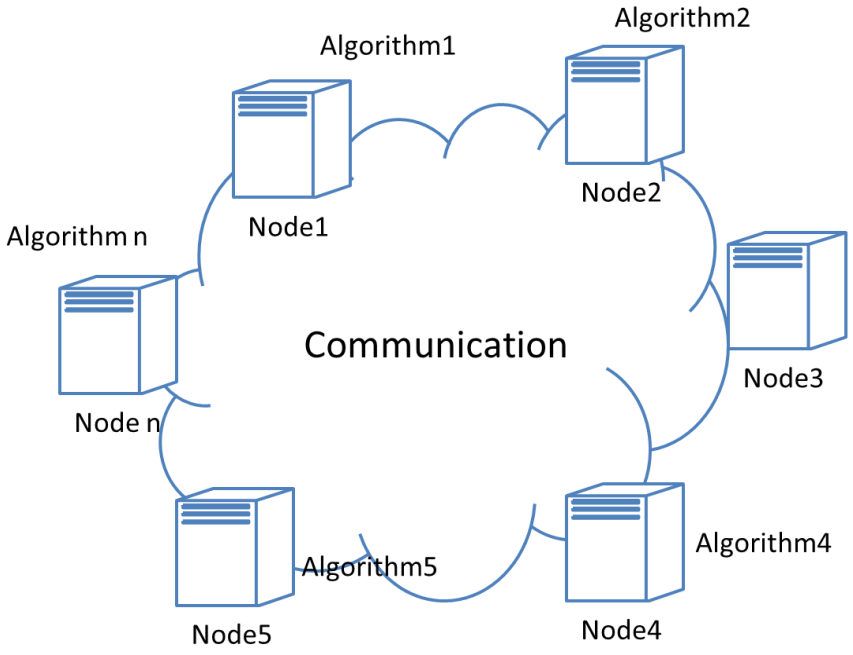

Figure 10. A computer cluster

4.1 Multi-Algorithm Communication Structure

4.1.1 Master-slave Structure

In the master-slave structure, a flexible global synchronizer is utilized. Each algorithm communicates with the global synchronizer rather than talks to each other in the simulation. The synchronizer broadcasts to inform all the algorithms the new solutions. The communication between the synchronizer and algorithm nodes is as Figure 11.

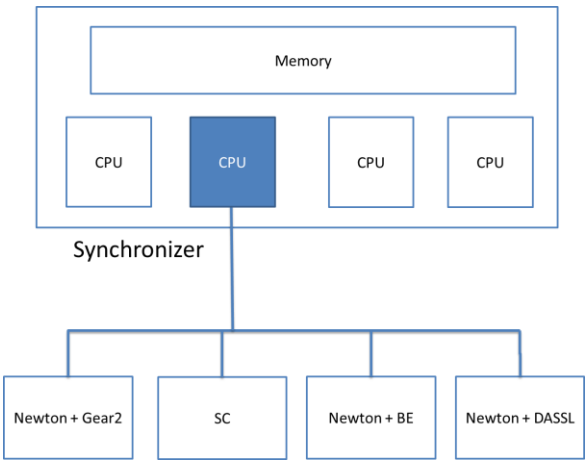

Figure 11. Global synchronizer node in Master-Slave structure.

In order to show a clear view of the hierarchy, we discuss the main roles that the algorithm node side and global synchronizer side play.

One algorithm node is demanded to send all circuit nodes' information including voltages or currents to the other algorithms to bring them to where it is standing. In addition, some algorithms like Gear2, DASSL, not only need the information at most recent time point, but also need several previous time steps solutions to calculate the new result. Hence, every algorithm sends k time steps results to the global synchronizer. Here,

k is determined by the highest order among the numerical integration methods in the system. From the foregoing discussion, Newton-Raphson needs previous one time step solution; Gear2 needs previous two time steps solutions while DASSL needs previous five time steps solutions. We keep k as 6 after taking the new solution into consideration.

In addition, an algorithm node fully controls granularity of the communication with the global synchronizer. In this implementation, we choose the granularity as one time step for all the algorithms. Hence, the algorithm node signals a communication thread to transfer the solution after it finishes one time step computation. The reason of creating a new thread to take over the interaction task is to overcome the coupling between communication and computation. Although the algorithm node can use the non-blocking MPI send method to transfer its own solutions, the MPI broadcast method in receiving the most recent solutions back is blocking. Figure 12 shows a computer node with 4 cores on which the BE + Newton is mapped.

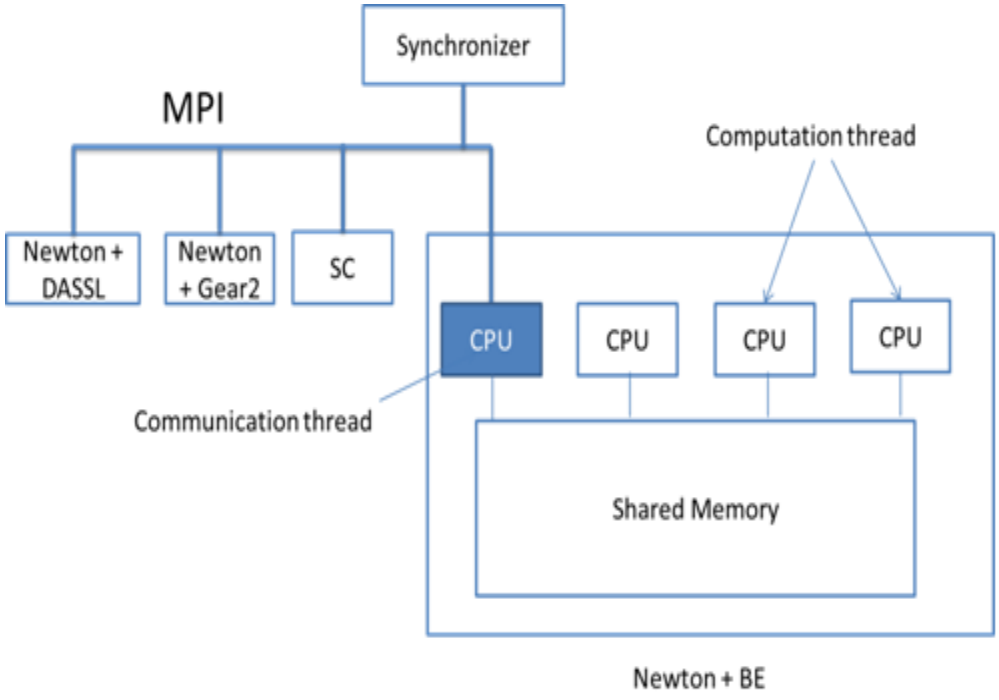

Figure 12. Details of an algorithm node in the Master-Slave structure.

Because the communication load in the global synchronizer is impressively large, the synchronizer is mapped to a single node to avoid memory contention. During simulation, it monitors all algorithm nodes. As soon as one algorithm node is sending a new solution, the synchronizer makes the connection and receives the solution.

The synchronizer maintains the most recent solution data structure the system has during the simulation. The data structure contains k time steps solutions. After the synchronizer receives a new message, the message is merge-sorted with the stored data, and the first k solutions are kept and the data structure is updated. If the new solution provided by an algorithm is ahead of the existing solutions, after merge sort, the data structure will be updated with the new solution by inserting it into the structure.

However, if the new solution is stale and lags behind the existing solutions, it will be abandoned and the solution structure stays unchanged.

After the global synchronizer processes one message and gets updated, it will broadcast new solutions to all algorithm nodes. Hence, all algorithms will be updated with the latest solutions and begin their next step calculation. In this way, the global synchronizer will always keep the most recent solutions and algorithm nodes interact with each other indirectly. The detailed work flow of the system is shown in Figure 13.

In the master-slave structure, all algorithms will be synchronized continuously. Slow execution of each of these algorithms is sidestepped by others and their advantages will be fully exploited. However, the global synchronizer needs to process and transfer a large amount of data since there are several nodes continuously sending messages to it. Consequently, the synchronizer may easily be the bottleneck of the system and affect system efficiency.

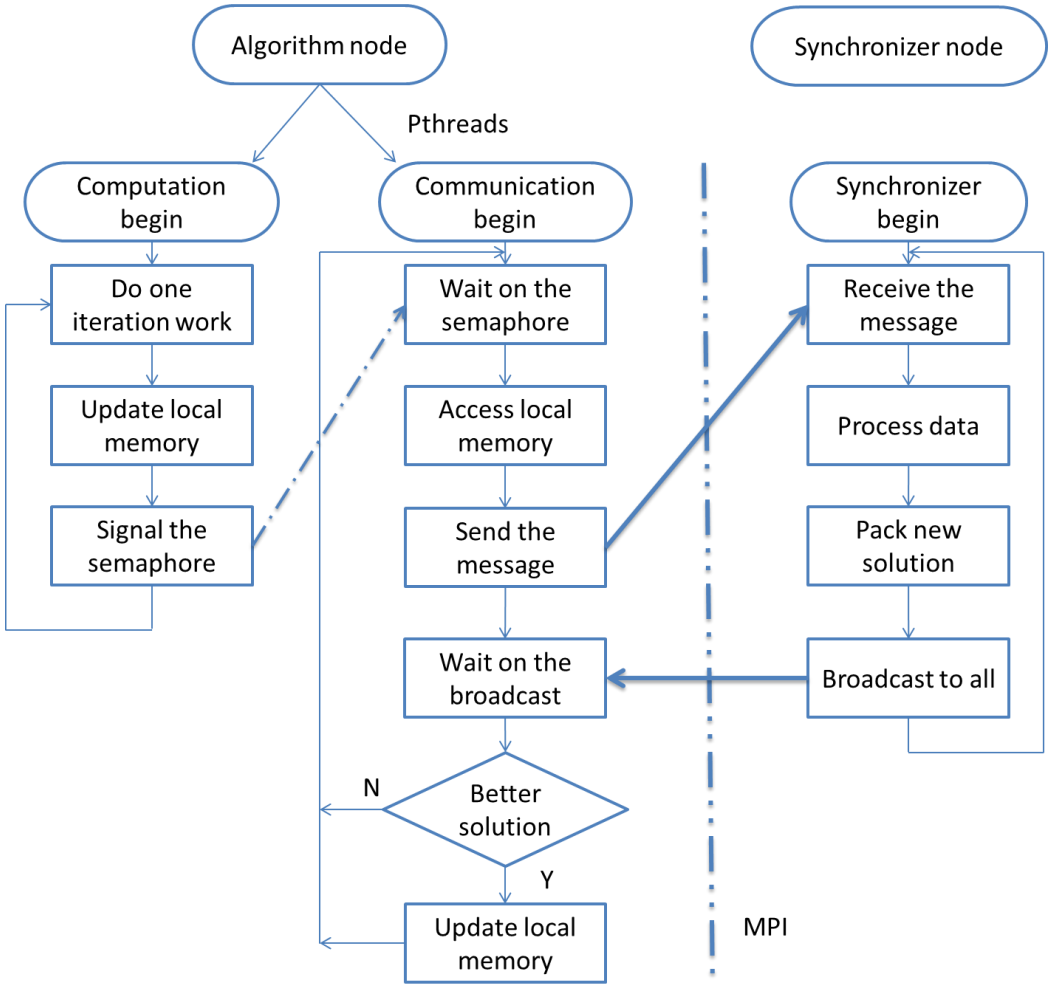

Figure 13. Flow chart of the algorithm node and global synchronizer in Master-Slave scheme.

4.1.2 Peer-to-Peer Structure

To avoid the bottleneck on the synchronizer, we come up with a peer-to-peer scheme. In this structure, one algorithm node similarly creates two threads for computation and communication, respectively. The communication thread receives messages from its preceding node, processes the received message with its own solutions, then sends the updated solution to the next node. The four algorithms form a loop and the most recent solutions keep circulating in the loop to synchronize all algorithms and explore their diversity. The communication structure is shown in Figure 14.

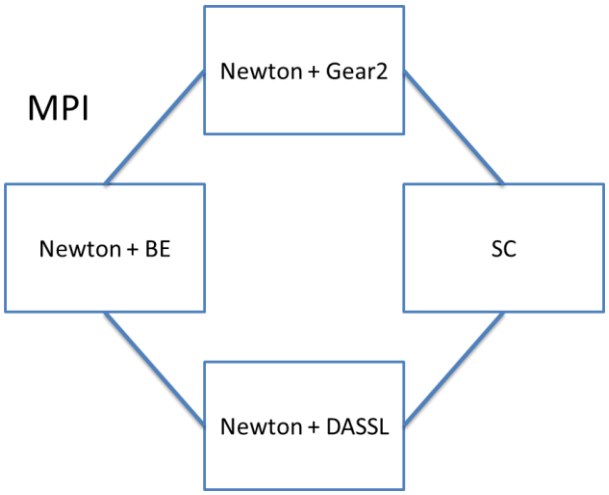

Figure 14. Peer-to-Peer communication scheme.

Apparently, this structure saves the resource by abandoning the global synchronizer and distributes the large amount of data processing work burden on the global synchronizer to each algorithm node. It eliminates the effect of bottleneck and also decreases the network load because in the master-slave structure the communication is collective and algorithm node may be not aware the status of the global synchronizer

and sends a stale solution which will occupy the network bandwidth and hamper effective communication. The main disadvantage is that in the peer-to-peer structure, all algorithms will be updated only when one-loop data transfer is completed. However, in the master-slave structure, all other algorithms will be informed immediately as soon as any one algorithm gets a new effective solution.

In this loop structure, deadlock, start and exit of the program needs additional attention. For instance, deadlocks happen when the successor node waits on a blocking MPI message from the precursor node which has reached the end of the simulation and exited. In our implementation, algorithm BE + Newton which is the most stable and has low computational cost for the initial time steps is used to trigger the transfer of data as a loop. At the end of the simulation, a flag is used to track how many nodes have finished. Every node will increment the flag before it exits. The flag is stored in the MPI message. Hence, when a node receives a message with a flag value equal to the number of all other algorithms, it knows all previous nodes have finished and it skips sending the message to the next node and exits. This way, the system can exit correctly. Figure 15 shows the detailed work flow in this structure.

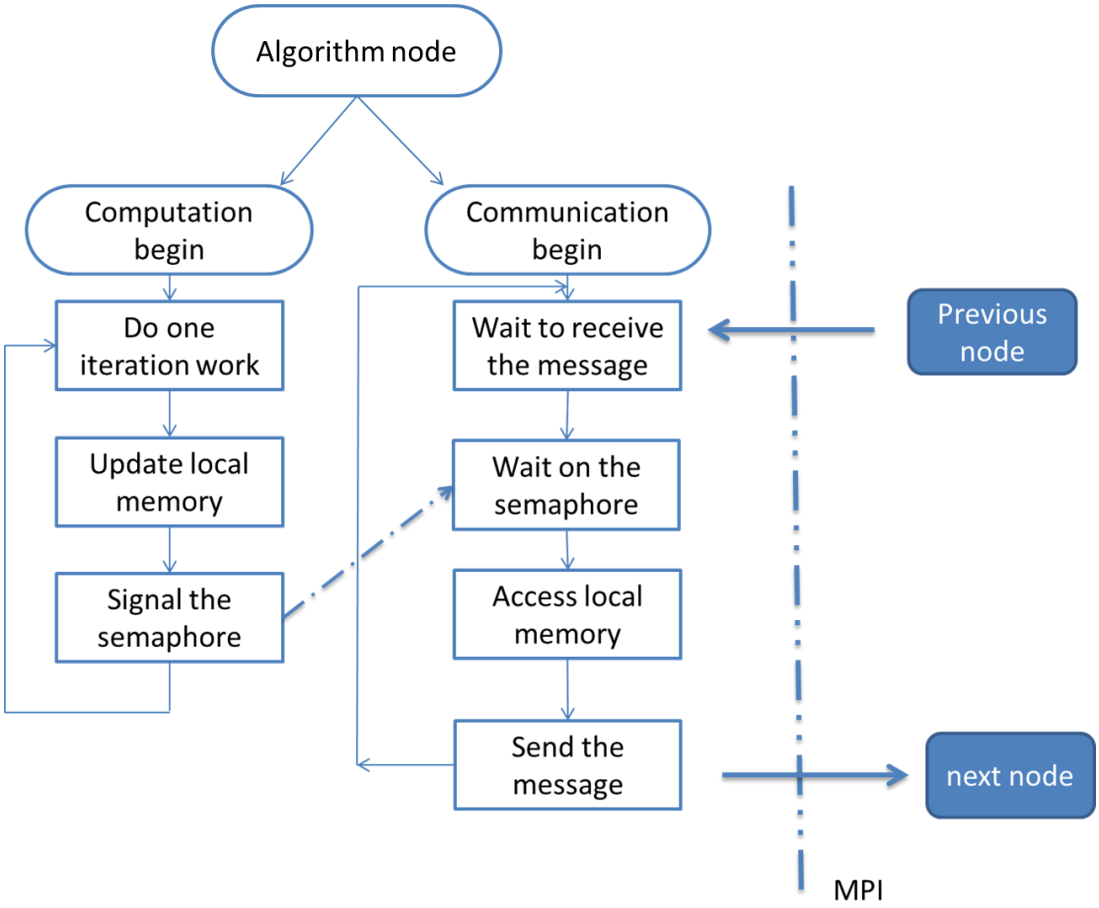

Figure 15. Flow chart of the algorithm node in peer to peer scheme.

4.2 Multiple Threads in A Single Algorithm

Transient analysis may be conducted over a large number of time steps. At every time step, it needs several iterations to get convergence. Hence, the number of iterations can be very high. Device evaluation and matrix solve carried on at every iteration are very time consuming and take nearly the whole simulation time. In previous discussion, there is a tradeoff between the number of the iterations per time step and the cost of each iteration for different nonlinear iterative methods. Here we further made use of the power of multi-core processor to expedite the device evaluation and matrix solve in a single algorithm node. A distributed platform provides the possibility of fully realizing intra-algorithm parallelism as one algorithm mapped on one node can exclusively access all the compute and memory resources.

4.2.1 Parallel Device Evaluation

In the device evaluation, Jacobian matrix $J(x_k)$ has a large number of partial differential items. In parallelization, nonlinear elements are divided into several groups, and each group is handled by one thread. The speedup for this can reach linear scaling when there are sufficient nonlinear elements. However, because of the cost of spawning, execution and termination of threads, the benefits of parallelization may be reduced especially when nonlinear elements in the circuit are few.

4.2.2 Parallel Matrix Solver

In our platform, SuperLU [17] is made use of as parallel matrix solver. SuperLU is a general purpose library providing direct solution to large, sparse, non-symmetric systems of linear equations on high performance machines. The library routines perform LU decomposition with partial pivoting and triangular system solves through forward and backward substitution. It exploits two sources of parallelism in the sparse LU factorization. The coarse level parallelism comes from the sparsity of the matrix, and is exposed by the column elimination tree of the matrix. The second level of parallelism comes from pipelining the computations of dependent columns.

The performance of matrix solve has bottleneck after the number of threads used reaches a certain number due to the circuit's and the computer node's characteristics. For instance, when using more threads in SuperLU, accessing critical sections via locks will increase and result in degradation of parallel performance. The more processors there are, the larger communication loss there will be. Second, the solver needs to divide the matrix into several parts and pipeline the operation on every part. Hence, the dense and small matrix generated by device evaluation has more dependence and is hard to be divided to several independent parts, making the parallel performance worse. On the contrary, the speedup is large for the sparse and large matrices.

The computer node on our platform is a symmetric multi-processor system with 8 dual core processors. The communication between the dual cores in one packaged processor chip is twice as faster as the communication between the cores in different processors chips. Hence, the performance of the parallel matrix solve has a degradation

when the number of the cores reaches to an odd number since the new added core needs to transfer data to cores in other chips. We choose to use even number of threads for parallel device evaluation and matrix solve which achieve better speedups.

5. RESULT AND ANALYSIS

5.1 Supercomputer

Hydra (see Figure 16) is a 52-node, 832-processor IBM cluster. The 52 nodes are further organized and housed into five physical frames [18]. The cluster uses IBM high-performance communication switch for parallel processing and other communication between the nodes. Each node connects to the HPS network using two adapters. HPS routes a message packet to another node [18].

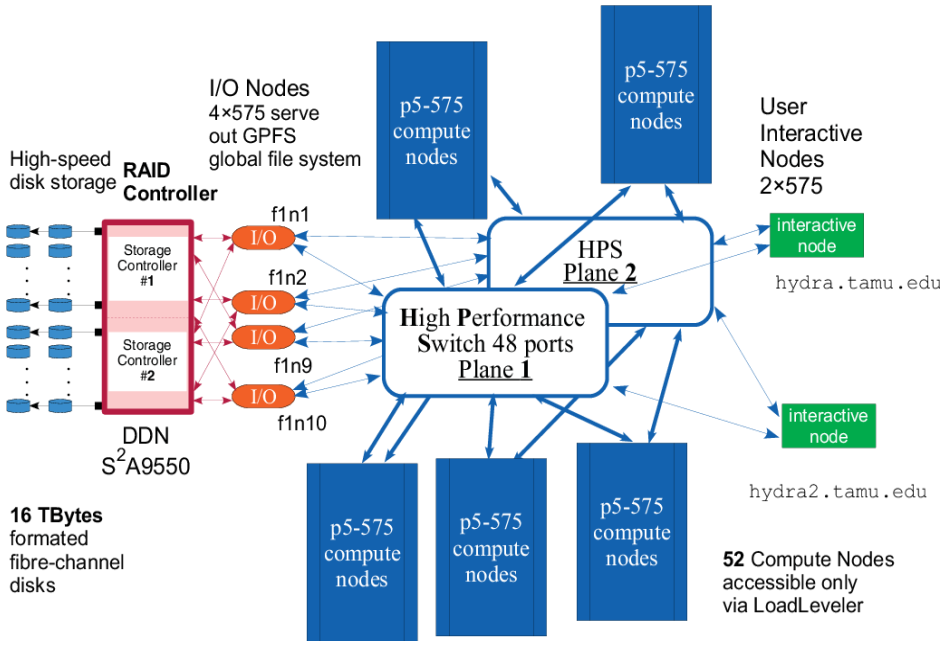

Figure 16. A snapshot of the supercomputer Hydra.

On Hyrda, when running a Pthreads program, the number of threads during execution can be set by the environment variable OMP_SET_NUM_THREADS. An MPI program is executed under the Parallel Operating Environment (POE). When the

program is being executed, the number of tasks can be set by the environment variable PROCS. Typically, tasks are mapped 1-to-1 on processors.

In the batch file, we can specify how tasks to be assigned. We assign the MPI tasks to 5 nodes with variable *node*. Every node can use 4 CPUs and 1.5gb memory by setting *ConsumableCpus as 4, ConsumableMemory as 1500mb* where 1500mb is the aggregate amount of memory taken up by 4 threads.

5.2 Result

5.2.1 MPI vs. Sequential Algorithm

First, we compare the MPI master-slave (MPI-MS) structure's runtime results with the four single sequential algorithms: Newton+BE, SC, Newton+Gear2, Newton+DASSL for several circuits in Table 1. The runtime results are in seconds. MPI-MS 1 core means that we use one core for one algorithm in the system. The speedup1 is MPI-MS 1 core over Newton + BE, which is the basic SPICE setup. MPI-MS 2 cores is that we assign 2 cores for every algorithm. The speedup2 is its speedup over MPI-MS with 1 core. The "N/A" in the table means the algorithms are not stable or diverge in the simulation.

Table 1. Comparison between sequential algorithm and MPI methods

	size /MB	No. of Lin. ele.	No. of FETs	No. of nodes	Newton BE/s	SC/s	Newton Gear2/s	Newton DASSL/s	MPI-MS 1 core/s	speed up1	MPI-MS 2 cores/s	speed up2
mesh4	100	3500	20	3500	178	53.5	78.5	68	52	3.42	49	1.06
mesh6	385	8500	40	8500	3960	86.2	1644	N/A	79	50.13	72	1.10
mesh18k	420	10k	50	10k	11060	213	4800	N/A	194	57.01	189	1.03
mesh28k	772	15k	50	15k	31000	429	12800	N/A	412	75.24	411	1.00
inv_chain1	71	1000	2000	1000	2311	632	826	431	395	5.85	205	1.93
inv_chain2	121	2000	4000	2000	2181	437	1040	491	428	5.10	240	1.78
grid20k	300	12k	0	8000	282	190	116	182	92	3.07	89	1.03
grid30k	600	18k	0	12k	441	316	180	285	150	2.94	135	1.11
4b_adder	22	50	200	250	140	749	74	106	60	2.33	32	1.88
lna_mixer	23	50	10	50	74	N/A	21	35	20	3.70	24	0.83
mixer	23	20	10	30	77	N/A	28	60	26	2.96	31	0.84

For mesh circuits [19], which have lots of linear elements and few nonlinear transistors, SC method is the fastest algorithm by avoiding repeatedly evaluating devices and factorizing large matrix. It can get convergence at every time point quickly. Compared to SC method, other algorithms cannot save this large amount of time and needs longer time to finish the simulation. This situation is more obvious for larger mesh circuits like mesh18k, and mesh28k which takes BE + Newton algorithm several hours to complete. MPI master-slave structure takes advantage of SC method and reaches a significant large speedup over Newton + BE.

The invert-chain circuits have more nonlinear elements. SC algorithm demands a lot of iterations to get convergence due to more complicated circuit operating condition and its worse convergence rate. In this case, the number of iterations dominates the cost for each time step even the cost for one iteration is still small. The multi-step integration methods perform better in these circuits especially when the circuits are small. The MPI master-slave structure which exploits the diversity of different algorithms and the advantages of different algorithms in different stages, reaches the smallest simulation time.

Mixer circuits are one kind of analog circuits with small size, high accuracy requirements and complex transistor operating condition changes. SC algorithm may not get convergence for whole simulation time. The Newton + Gear2 algorithm is getting results fast. The MPI master-slave method can run a little faster than Newton + Gear2 with other algorithms' contributions.

After applying more threads in single algorithm in the distributed system, we find that speedup2 almost reaches the optimal for the inverter chain circuits. This may be due to the fact that the inverter chain circuits consist of a large number of transistors which can be divided equally into two groups and handled efficiently by two threads. In addition, the size of the matrix obtained by device evaluation is suitable for the parallel matrix solver. The speedup for other circuits is not as good as inverter chains. Even worse, analog circuits have performance drop after being applied two threads for a single algorithm. Analog circuits are either small or with a small number of nonlinear elements and have large overhead in parallel device evaluation and matrix solve. Creating/terminating threads introduces a relatively larger cost to these small circuits. The benefits introduced by multiple threads are smaller than the overhead.

These results demonstrate the benefits brought by the MPI based multi-algorithm circuit simulation and multiple threads in a single algorithm for certain classes of circuits.

5.2.2 MPI vs. HMAPS

In this section, the results between HMAPS [20] and MPI based distributed simulation are compared. HMAPS run in one node with 8 threads and 2 gigabytes memory while MPI methods are using two threads for each algorithm on several nodes. The resource allocation and results are in Table 2 and Table 3. The size/MB column shows the memory size of one circuit data copy. Column HMAPS, MPI-MS and MPI-P2P show the runtimes in second. The MPI-MS speedup is the MPI master-slave structure's

speedup over HMAPS while the MPI-P2P speedup is the MPI peer-to-peer structure's speedup over HMAPS.

Table 2. Resource allocation between HMAPS and MPI methods.

	Threads/algorithm	Nodes	Threads/node	Memory
HMAPS	2	1	8	2GB
MPI master-slave	2	5	3	2GB
MPI Peer-to-Peer	2	4	3	2GB

Table 3. Comparison between HMAPS and MPI methods.

Circuit	size/MB	HMAPS/s	MPI-MS/s	MPI-MS speedup	MPI-P2P/s	MPI-P2P speedup
mesh4	100	56	49	1.14	51	1.10
mesh6	385	85	72	1.18	74	1.15
mesh8	420	226	189	1.20	196	1.15
mesh18	772	750	411	1.82	379	1.98
inv_chain1	71	236	205	1.15	214	1.10
inv_chain2	121	249	240	1.04	230	1.08
grid20k	300	107	89	1.20	92	1.16
grid30k	600	210	135	1.56	124	1.69
4b_adder	22	46	32	1.44	35	1.31
lna_mixer	23	42	24	1.75	25	1.68
mixer	23	81	31	2.61	32	2.53

In HMAPS [20], multiple algorithms are mapped to a single shared-memory system and every algorithm shares computing resources. In the results above, four algorithms are running with their own copy of circuit data, with totally four copies on one computer node. It requests 3 gigabytes for the mesh18 circuit and 2.5 gigabytes memory for grid30k. On the 2 gigabytes shared-memory system, the memory contention is large and the simulation takes longer time to finish. The MPI based distributed system runs algorithms on separate nodes. The memory used on one node is 800 megabytes for mesh18 circuit and 600 megabytes memory for grid30k. Hence, the memory contention is smaller and speedup can reach as high as 1.98.

The MPI based methods are about 15 percentages faster for mesh4, mesh6, grid20k and inverter chain circuits. These circuits normally need about several hundred megabytes memory but MPI structures have more communication overhead than HMAPS where threads access shared local memory quickly and the communication between the algorithms can be made frequent. In the distributed system, communication speed is limited by the network bandwidth and the size of messages. The communication cost and delay could be large when simulating large circuits. However, the MPI based platform is capable of incorporating more algorithms to further exploit inter-algorithm parallelism which is more difficult for HMAPS.

5.2.3 Comparison Between MPI Methods

The comparison between the MPI master-slave structure and the peer-to-peer structure is shown in Figure 17. The speedups are the two MPI based methods' speedups over HMAPS.

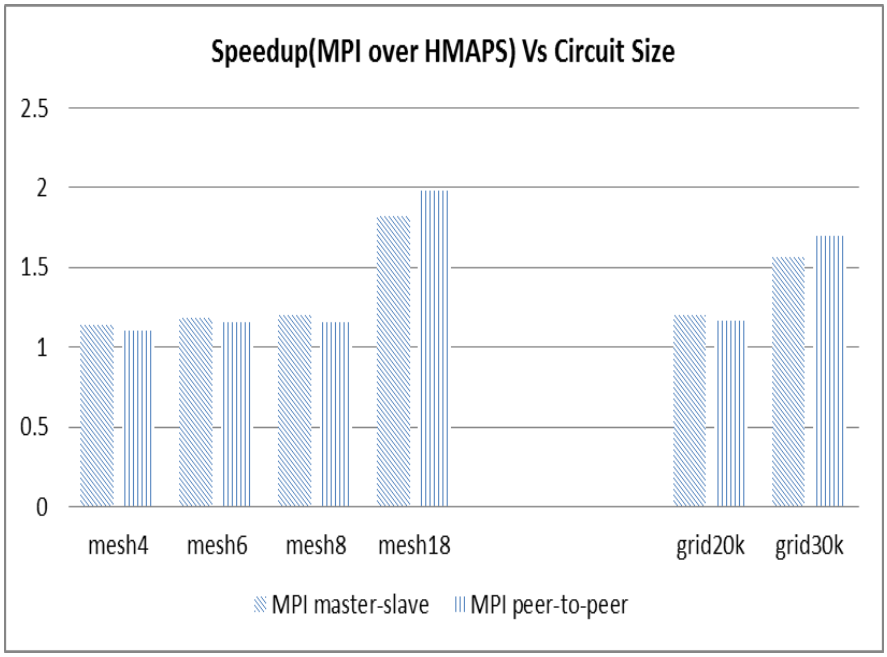

Figure 17. Comparison between master-slave and peer-to-peer communication structure.

For small circuits like mesh4, mesh6, mesh8 and grid20k, each algorithm updates the global synchronizer quickly after getting its own solution in the MPI master-slave structure. The synchronizer will also broadcast and inform every algorithm the most recent solution immediately. It has little bottleneck due to the fact that the circuit size is small and the data processing is quick. However, the MPI peer-to-peer structure has a delay in updating all algorithms because the algorithms receive the latest solution only after the solution experiences one loop transfer. This is demonstrated in the figure above

which show that the MPI master-slave structure is faster than the MPI peer-to-peer structure.

For large circuits, like mesh18 and grid30k, the speedup of the MPI master-slave scheme is much smaller than the MPI peer-to-peer scheme. In these circuits, the messages generated by the circuit have huge size and the synchronizer needs to receive a large amount data from the algorithm nodes during the simulation as well as the data processing time in synchronizer is increasing. These factors put a large work load on the global synchronizer and cause a bottleneck. Moreover, the algorithms may send the stale solutions to the synchronizer because they are not aware of the status of the synchronizer. In this case, the network bandwidth is occupied and wasted by these kinds of useless communication. In the peer-to-peer scheme, the processing and network load is distributed among all the algorithm nodes and the bottleneck effect is alleviated. In addition, one node resource which is occupied by the synchronizer is saved.

5.2.4 Accuracy

We compare the results between BE + Newton and the distributed circuit simulation on two nodes of mesh4 circuit in Figure 18 and Figure 19. The BE + Newton is the basic SPICE setup and accurate. We compare the two voltages from the two methods on the same time points, and the standard deviation is smaller than 0.001 volt. Hence, the simulation results are acceptable.

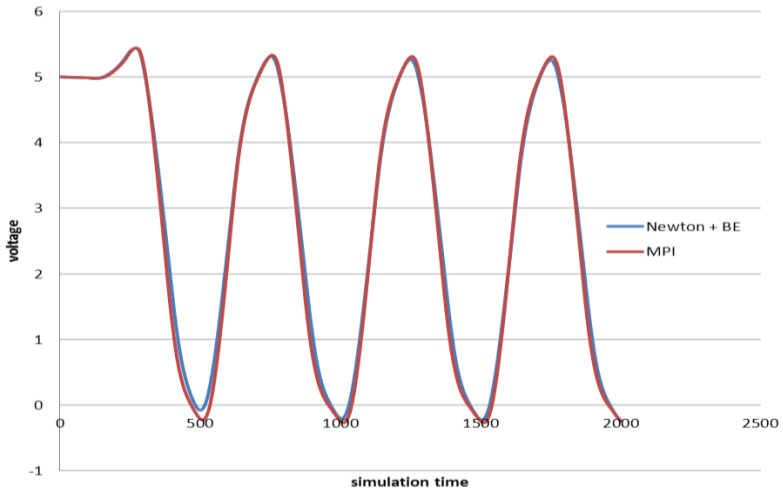

Figure 18. Simulation results on Node1

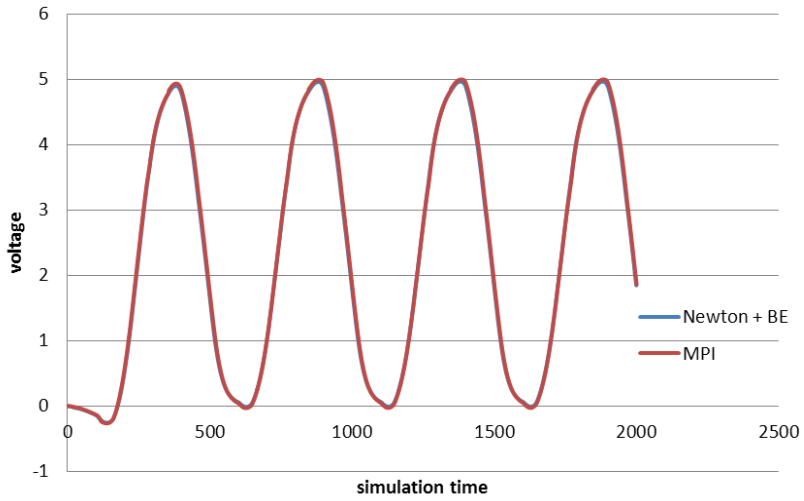

Figure 19. Simulation results on Node2

6. CONCLUSIONS

In this thesis work, we exploited parallel and distributed multi-algorithm circuit simulation to alleviate the significant computational costs of parallel circuit simulation on shared memory systems. We exploited inter-algorithm parallelism by developing the master-slave structure and peer-to-peer structure and introducing intra-algorithm parallelism through parallel device evaluation and matrix solve to improve runtime efficiency. With the above techniques, our experimental results demonstrate that MPI based distributed platform decreases memory contention and thread contention, reaches a large speedup and is capable for incorporating more algorithms into the system.

REFERENCES

[1] Electronic circuit simulation [online]. Available: http://en.wikipedia.org/wiki/Electronic_circuit_simulation. Accessed on Mar. 2012.

[2] G.Yang, “Paraspice: A parallel circuit simulator for shared-memory multiprocessor,” in *Proc. ACM/IEEE Design Autom. Conf*., pp. 400-405, Jun. 1991.

[3] N. Rabbat, A. Sangiovanni-Vincentelli, and H. Hsieh, “A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain,” in *IEEE Trans. Circuits Syst.,* vol. 26, no. 9, pp. 733–741, Sep. 1979.

[4] W. Dong, P. Li, and X. Ye, "WavePipe: parallel transient simulation of analog and digital circuits on multi-core shared memory machines", in *Proc. of IEEE/ACM Design Automation Conference (DAC),* pp. 238-243, Jun. 2008.

[5] N. Rabbat, A. Sangiovanni-Vincentelli, and H. Hsieh, “A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain,” *IEEE Trans. Circuits Syst.*, vol. 26, no. 9, pp. 733–741, Sep. 1979.

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

[7] X. Ye, W. Dong, P. Li, S. Nassif, "MAPS: Multi-algorithm Parallel Circuit Simulation," in *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD),*pp. 73-78, Nov. 2008.

[8] P. Li, "Parallel Circuit Simulation: A Historical Perspective and Recent Developments," in *Foundations and Trends in Electronic Design Automation,* vol. 5, no 4, pp 211-318, 2011.

[9] A. Jameson and E. Turkel, "Implicit Schemes and LU Decompositions," in *ICASE Report*, pp. 79-24, 1979.

[10] Pthreads [Online]. Available: http://en.wikipedia.org/wiki/Pthreads. Accessed on Mar. 2012.

[11] Message Passing interface [Online]. Available: http://en.wikipedia.org/wiki/ Message_Passing_Interface. Accesssed on Mar. 2012.

[12] W. Nagel, and O. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," memorandum no. ERL-M382, University of California, Berkeley, Apr. 1973.

[13] P. Li and L. Pilleggi, "A linear-centric modeling approach to harmonic balance analysis," in *Proc. Design Autom. Test Europe*, pp. 634–639, Mar. 2002.

[14] K. C. Yeh and K. C. Kwan, "A comparison of numerical integrating logarithms by trapezoidal, Lagrange, and spline approximation," *in J. Pharmacokin,* Biopharm. 6, pp. 79-98, 1978.

[15] W. Gear, "The numerical integration of ordinary differential equations," in *Math. Comp*., pp. 146-156, Apr. 1967.

[16] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solutions of Initial-Value Problems in Differential-Algebraic Equations*. New York: Elsevier, 1989.

[17] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal

algorithm for sparse Gaussian elimination," in *SIAM J. Matrix Anal.* Appl., vol 20, no. 4, pp. 915–952, 1999.

[18] Hydra Super Computer [Online]. Available: http://sc.tamu.edu/help/hydra/sysinfo.php. Accessed on Mar. 2012.

[19] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, and K. A. Jenkins, "A clock distribution network for microprocessors," in *IEEE J. Solid-State Circuits,* vol. 36, no. 5, pp. 792–799, May 2001.

[20] X. Ye, W. Dong, P. Li, and S. Nassif, "Hierarchical Multialgorithm Parallel Circuit Simulation," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* volume 30, pp. 45-58, January 2011.

VITA

Name: Ruicheng Dai

Address: 214 Zachry Engineering Center,
TAMU 3128
College Station, TX 77843-3128

Email Address: dairc2009@gmail.com

Education: B.S., Telecommunication Engineering,
Zhejiang University, China, 2009
M.S., Computer Engineering, Texas A&M
University, 2012