

Copyright © 1982, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

LSI CIRCUIT SIMULATION  
ON VECTOR COMPUTERS

by

A. Vladimirescu

Memorandum No. UCB/ERL M82/75

22 October 1982

**LSI CIRCUIT SIMULATION ON VECTOR COMPUTERS**

**by**

**Andrei Vladimirescu**

**Memorandum No. UCB/ERL M82/75**

**22 October 1982**

**ELECTRONICS RESEARCH LABORATORY**

**College of Engineering  
University of California, Berkeley  
94720**

## **LSI Circuit Simulation on Vector Computers**

**Andrei Vladimirescu**

### **ABSTRACT**

The simulation of Large-Scale-Integrated (LSI) circuits requires very long run times on conventional circuit analysis programs such as SPICE2. A new simulator for LSI circuits, CLASSIE, has been developed which is more efficient and preserves the same accuracy.

Two basic factors of present technology are considered in the design of the new LSI circuit simulator. First, LSI circuits are usually a collection of a limited number of structurally identical functional blocks such as logic gates, operational amplifiers, etc. The second is the availability of vector computers which provide an ideal architecture for fast computations on repetitive structures. SPICE2 operates on an entire circuit matrix which is processed at the individual electrical element level. The analysis in the new program takes into consideration the structure of the LSI circuit. The identical functional blocks are grouped together and the simulation is performed at two levels.

The above design considerations speed up the simulation of an LSI circuit performed by CLASSIE considerably compared to SPICE2. For the analysis of a large circuit, CLASSIE on a vector computer rates in simulation speed between SPICE2 and a timing simulator.

For various test circuits containing from a few hundred to a few thousand semiconductor devices, CLASSIE simulation runs on a CRAY-1

indicate that the speedup compared to SPICE2 increases with circuit size. The solution techniques implemented in the new simulator make the execution time grow linearly with increasing circuit complexity in comparison to an exponential growth in a conventional circuit simulator. Vectorized device-model evaluation and a machine-code solver using both vector and scalar operations bring about the increased performance.

A program such as CLASSIE creates a framework for hierarchical circuit simulation. The decoupling of the analysis at the subcircuit (cell) level allows the implementation of direct solution algorithms, as in CLASSIE, as well as indirect (relaxation-type) solution algorithms in future programs. Further modifications can be made for optimal performance on various computer architectures (single-instruction single-data stream, single-instruction multiple-data stream, and multiple-instruction multiple data stream).

## ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor D. O. Pederson who has given me the opportunity to work in his research group and has guided and supported me throughout the course of this project.

The common work with E. Cohen has been of significance to the success of this research. The close working contacts with A. R. Newton, A. Sangiovanni-Vincentelli, D. A. Calahan, R. Dowell, L. Nagel, G. Boyle, S. Liu, and J. Kleckner have contributed useful ideas. The discussions with J. Crawford, A. Lachner, T. Quarles and the other members of the Computer-Aided Design research group are also acknowledged.

The present project has been made possible by a grant from the Bell Laboratories. The active support received from G. Smith, H. Boll, and H. Gummel and the close working relation with Bell Laboratories researchers have significantly contributed to the successful completion of this research.

The encouragements of W. G. Oldham, D. J. Angelakos, D. A. Hodges, and P. R. Gray are greatly appreciated. The staff of the Electronics Research Laboratory and Graduate Office of the EECS have provided a most cooperative environment.

The guidance of C. Bulucea, M. Bodea, and G. Smarandoiu during my first years as a researcher in Romania was important in my early professional efforts.

A special dedication is made to my late mother Jeana-Maria-Victoria who gave me the love and confidence to achieve the present performance. I

**thank my father, Gheorghe, as well as relatives and friends from nearby and far away for their understanding and encouragement.**

## TABLE OF CONTENTS

<b>CHAPTER 1: Introduction .....</b>	<b>1</b>
<b>CHAPTER 2: Accuracy Issues in Circuit Simulation .....</b>	<b>4</b>
2.1 Introduction.....	4
2.2 Algorithms in Standard Circuit Simulation .....	5
2.3 A State-of-the-Art in Simulation .....	10
2.3.1 Algorithms for Large Circuit Simulation.....	11
2.3.1.1. Tearing Decomposition.....	12
2.3.1.2. Relaxation Decomposition .....	19
2.3.2 Computers and Simulation for LSI Circuits .....	21
2.3.2.1. Computer Hardware.....	21
2.3.2.2. Vector Processor Performance Evaluation .....	26
2.3.2.3. Algorithm Evaluation on Vector Processors.....	30
2.4 Semiconductor Device Modelling.....	34
2.5 Convergence.....	38
2.6 Linear Equation Reordering and Solution .....	41
<b>CHAPTER 3: CLASSIE - Design Considerations .....</b>	<b>45</b>
3.1 Introduction.....	45
3.2 Circuit Simulator Performance and Computer Hardware .....	46
3.3 Integrated Circuit Specifics and Representation .....	56
3.4 Algorithms and Implementation .....	59
3.4.1 Modified Nodal Admittance Matrix.....	59
3.4.2 Semiconductor Device Model Evaluation.....	62
3.4.3 Memory Access Considerations .....	67
<b>CHAPTER 4: Program Description .....</b>	<b>70</b>

4.1 Introduction.....	70
4.2 Data Structures.....	71
4.2.1 Description of Data Structures .....	71
4.2.2 Memory Manager Functions.....	77
4.3 Newton-Raphson Scheme Implementation .....	79
4.4 Vectorized Semiconductor Device Model Evaluation .....	86
4.5 Linear Equation Solution.....	91
4.5.1 Processing in the Setup Phase.....	92
4.5.2 Reordering .....	94
4.5.2.1 Interconnection Matrix .....	94
4.5.2.2 Subcircuit Matrix .....	97
4.5.3 Machine Code Solution .....	98
<b>CHAPTER 5: Performance Evaluation of CLASSIE .....</b>	<b>106</b>
5.1 Introduction.....	106
5.2 Speed-Performance Measurement.....	106
5.3 Benchmark Presentation.....	108
5.4 SPICEV Performance .....	110
5.5 CLASSIE Performance.....	117
5.6 Simulation Time Breakdown.....	128
5.7 Small-Circuit Performance.....	133
<b>CHAPTER 6: Conclusion.....</b>	<b>139</b>
6.1 Hierarchical Simulation .....	140
6.2 SIMD Processor Role .....	141
6.3 Salient Results and Future Perspective.....	143
<b>APPENDIX 1: SPICE 2G Small-Size MOSFET Charge Model.....</b>	<b>146</b>
A1.1 Small-Size MOSFET Model Considerations in SPICE2 .....	146
A1.2 Charge Formulations.....	148

<b>APPENDIX 2:</b>	<b>Benchmark Circuits .....</b>	<b>153</b>
<b>APPENDIX 3:</b>	<b>Subcircuit Definition Table Specification .....</b>	<b>156</b>
<b>APPENDIX 4:</b>	<b>Adder4 and Adder16 Sample CLASSIE Simulations.....</b>	<b>158</b>
<b>APPENDIX 5:</b>	<b>Small-Circuit CLASSIE Simulations .....</b>	<b>160</b>
<b>APPENDIX 6:</b>	<b>CLASSIE Program Listing .....</b>	<b>162</b>
<b>REFERENCES</b> .....		<b>164</b>

## CHAPTER 1

### INTRODUCTION

The SPICE program has had wide acceptance and use for integrated circuit evaluation during the last ten years. Although initially designed to simulate efficiently circuits containing up to one to two hundred elements, SPICE 2G is presently used to analyze circuits which are one to two orders of magnitude larger. State-of-the-art scientific computers such as the CRAY-1 perform a SPICE2 simulation of a large circuit much faster than a more common super-mini computer such as the VAX 11/780. In spite of an inherent increase in the execution speed of SPICE2 due to the hardware, an additional order of magnitude increase in speed is needed for the efficient simulation of LSI circuits.

This dissertation presents CLASSIE, a new simulator for LSI circuits, developed as part of the requirement for the doctoral degree at the University of California, Berkeley. Two basic factors of present technology are considered in the design of the new LSI circuit simulator. The first one is that LSI circuits are usually a collection of a limited number of structurally identical functional blocks such as logic gates, operational amplifiers, memory cells, etc. The second factor is the advent of vector computers which provide an ideal architecture for fast computations on repetitive structures. SPICE2 operates on an entire circuit matrix which is processed at the individual electric element level. The analysis in the new program takes into consideration the structure of the LSI circuit. The identical

functional blocks are grouped together and the simulation is performed at two levels. Due to this approach to circuit simulation an LSI circuit can be analyzed in a time interval which is half way between the time needed on the same computer, by a conventional circuit simulator and a timing simulator

The second chapter contains an overview of the algorithms used in 'standard' or 'second-generation' circuit simulation contrasted with the techniques employed in 'third-generation' simulators developed presently for large circuits. The computer hardware available for the new simulators is also described with emphasis on the architectural features, performance evaluation and algorithm implementation. A summary of the modeling issues, convergence and linear-equation solution as implemented in the latest version of SPICE2 form the rest of the second chapter.

Chapter 3 presents the design considerations and decisions for the new LSI simulator. For a better understanding of the impact of different architectures on the performance of SPICE2, run time statistics are compared. The specifics of large circuits and their simulator implementation is described followed by an outline of algorithm peculiarities for the CRAY-1 vector processor.

Chapter 4 contains a detailed presentation of the program. The data structure is examined in close relation to the memory management package which supports it. The implementation of the Newton-Raphson iterative scheme in the two-level analysis and the fast semiconductor device model evaluation using vector computation are described in detail. The importance of numerical pivoting for the accurate solution of the subcircuit and interconnection equations, the linkage between the different matrices and the generated vector and scalar machine code are presented in the last part

of this chapter.

Two different programs have been used in the study of the speedup which can be obtained in circuit simulation using vector processors. The first, SPICEV, is a vectorized SPICE2 version. The major difference between the two programs is that the former groups semiconductor devices together and uses vector computation in device evaluation and other device-related computation, and scalar machine code for the linear-equation solution. The second, CLASSIE, differs from SPICE2 in the implementation of the hierarchical analysis, data structure and some solution algorithms. The performance of these three programs is compared on a number of large, medium and small (standard SPICE2 benchmarks) circuits in Chapter 5.

The conclusions on hierarchical circuit simulation and vector processing are presented in Chapter 6. A series of comments is made on the best implementation of a CLASSIE-type simulator on other vector and array processors.

## CHAPTER 2

### Issues in Circuit Simulation

#### 2.1. Introduction

Several years ago it was believed that circuit simulation is economical only for individual cells and circuit blocks with a maximum of a few hundred devices. The prohibitively long run times for an LSI circuit simulation made the newly emerging timing and mixed-mode simulators [Newt78], [DeMa79] attractive for the characterization of LSI chips. The use of logic, timing and mixed-mode simulators is a valid approach for certain classes of circuits and systems, e.g., large custom logic chips or entire processors. Circuit simulation has, however, remained a reliable tool for advanced designs and therefore continuing research has taken place for faster simulation at the transistor level [Raba79], [Yang80], [Vlad81a], while maintaining accuracy. Technology trend-setting products, such as dynamic RAMs, are studied best with the help of circuit simulation.

A number of algorithms have proven well suited for the solution of the equations of electrical and integrated circuits and have been implemented in most circuit simulators in existence to date. All programs which use these same algorithms, viz., numerical integration, Newton-Raphson iterations, and Gaussian Elimination, are referred to as standard circuit simulators and are described briefly in the second section of this chapter. The distinctive characteristic of a circuit simulator compared to other electrical

simulators, viz. timing and mixed-mode, is the use of direct methods only in the solution process.

The characteristics of LSI circuits have initiated the search for better suited algorithms which preserve the same accuracy as in standard circuit simulation but achieve higher simulation speeds. The third section of this chapter is dedicated to these 'third-generation' simulation techniques and the importance of modern day computer hardware.

Semiconductor device modeling is a central issue in the simulation of integrated circuits. The present solutions to modeling are described in another section of this chapter.

Most important for converging to the correct solution for the simulated circuit is an algorithm which controls the change in the nonlinear element state variables from iteration to iteration. This algorithm is known as a limiting algorithm and determines the convergence features of the simulator. Convergence is the subject of the sixth section in this chapter.

The linear-equation solution is the last in the sequence of algorithms which together make a circuit simulator a useful tool. A few details of the implementation of algorithms for sparse linear equations in SPICE2 are presented in the last section of this chapter.

## **2.2. Algorithms in Standard Circuit Simulation**

Practically all the circuit simulators available today, SPICE2, [Nage75], [Coh76], [Vlad81b], ASPEC, [Jenk82], ADVICE, [Nage80], SLIC, [Kop75], ASTAP, [Week73], SCAMPER, [Agne80], use the same general techniques for solving the electric circuit problem. A numerical, implicit integration method transforms the nonlinear differential equations into nonlinear

algebraic equations which in turn are linearized using a modified Newton-Raphson iterative algorithm; Gaussian elimination and sparse matrix techniques provide a solution to the simultaneous linear equations. This sequence of algorithms was described in 1971 in an overview paper [McCa71] and is referred to in today's literature, [Hach81], as part of the 'standard' or 'second-generation' circuit simulator.

Figure 2.1 shows a general scheme used in simulators and highlights the important steps. The problem to be solved can be expressed generically as

$$\mathbf{F}(\dot{\mathbf{x}}, \mathbf{x}, t) = 0 \quad (2.1)$$

where  $\mathbf{x}(t)$  represents node voltages and currents in the case of the Modified Nodal Analysis (MNA) approach. The MNA is an extension of nodal analysis in that the node voltage equations are augmented by current equations for the voltage-defined elements [Ho75], [Nage71]. In Equation 2.1  $\dot{\mathbf{x}}$  can be eliminated and approximated after applying an a-stable, or stiffly-stable numerical multi-step integration formula [Chua75],

$$\dot{\mathbf{x}}_{n+1} = \sum_{i=0}^k a_i \mathbf{x}_{n-i} + h_{n+1} \sum_{i=-1}^k b_i f(\mathbf{x}_{n-i}, t_{n-i}) \quad (2.2)$$

where

$$h_{n+1} = t_{n+1} - t_n$$

is the variable time step at  $t_{n+1}$ . This reduces Equation 2.1 at each time point to a set of nonlinear algebraic equations expressed concisely as

$$\mathbf{f}(\mathbf{x}) = 0 \quad (2.3)$$

where  $\mathbf{x}$  is the unknown vector.

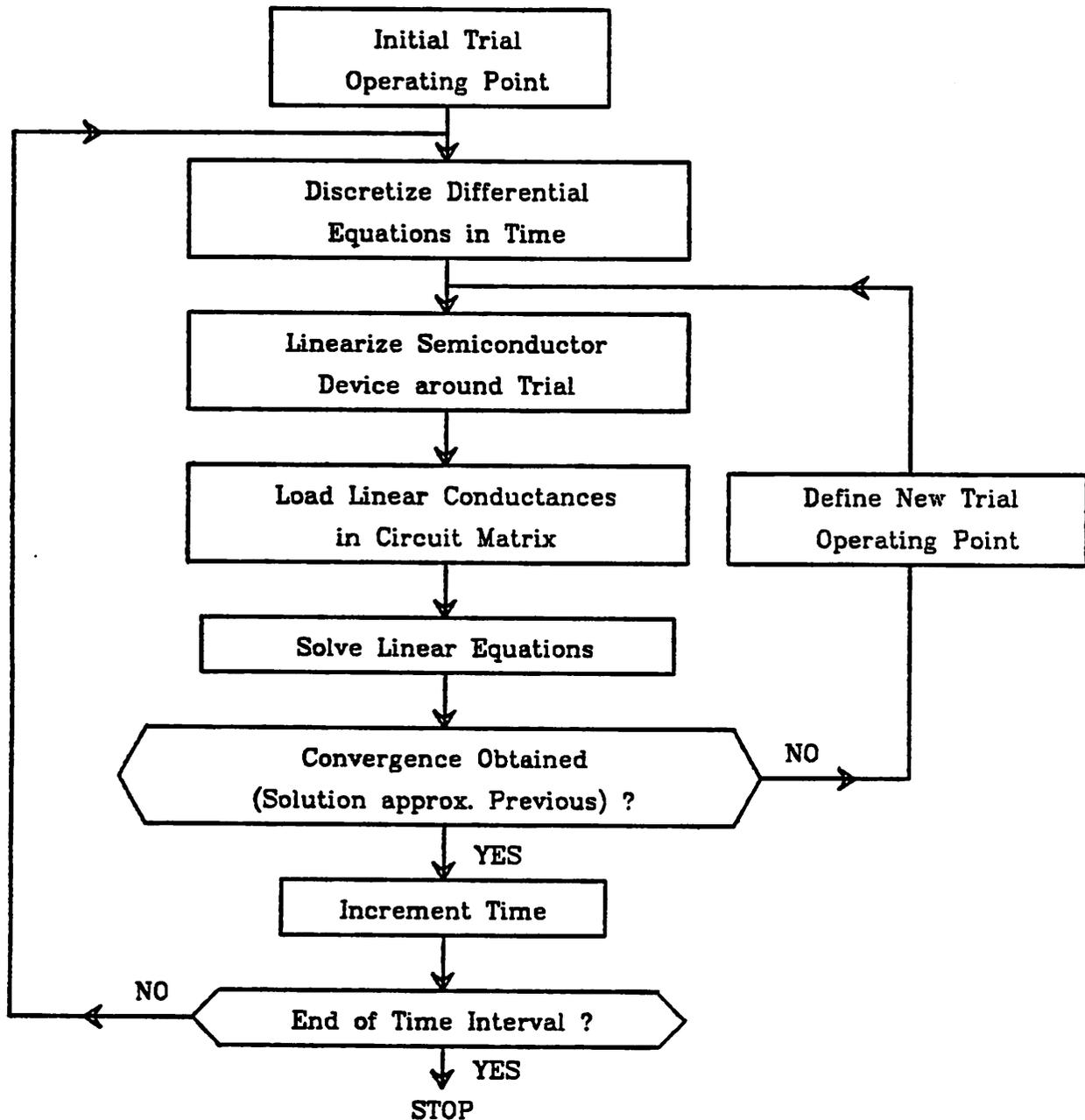


Figure 2.1. Numerical Integration and N-R Algorithm in Circuit Simulation

The simultaneous nonlinear equations 2.3 are solved using the modified Newton-Raphson method which can be expressed for the r-th variable as

$$\mathbf{x}_r^{m+1} = \mathbf{x}_r^m - \alpha(\mathbf{x}_r^m) \frac{g(\mathbf{x}_r^m)}{g'(\mathbf{x}_r^m)} \quad (2.4)$$

where  $\mathbf{x}^m$  is the approximation of the solution at the previous iteration and  $g(\mathbf{x})$  contains all nonlinearities. Parameter  $\alpha$ ,  $0 < \alpha \leq 1$ , indicates that only a fraction of the change is accepted at each iteration. The choice of  $\alpha$  is implemented through the limiting algorithm which is tailored to the different nonlinear characteristics of each semiconductor device. This parameter assumes a different value at each iteration and for each device.

Equation 2.4 replaces the nonlinear formulation in Equation 2.3 by a linear set of equations solved at each iteration. Equation 2.4 can be rewritten as

$$\mathbf{J}(\mathbf{x}^m) \mathbf{x}^{m+1} = \mathbf{J}(\mathbf{x}^m) \mathbf{x}^m - \mathbf{g}(\mathbf{x}^m) \quad (2.5)$$

where  $\mathbf{J}(\mathbf{x}^m)$  is the Jacobian computed with  $\mathbf{x}^m$  at the last iteration. The simultaneous linear equations which are solved using Gaussian elimination, have the matrix representation

$$\mathbf{A}\mathbf{x}^{m+1} = \mathbf{b} \quad (2.6)$$

where

$$\mathbf{A} = \mathbf{J}(\mathbf{x}^m) + \mathbf{G}$$

$$\mathbf{b} = \mathbf{J}(\mathbf{x}^m)\mathbf{x}^m - \mathbf{g}(\mathbf{x}^m) + \mathbf{C}$$

Equation 2.6 includes also the contributions from the linear elements in the circuit.

It can be seen that Equation 2.5 and Equation 2.6 can be rewritten to solve for  $\delta\mathbf{x}^{m+1} = \mathbf{x}^{m+1} - \mathbf{x}^m$ . For the first few iterations the two solution approaches generate almost the same result if an initial solution  $\mathbf{x}^0 = 0$  is

used. Once  $\mathbf{x}$  is close to the solution, solving for  $\delta\mathbf{x}$  provides more significant digits of accuracy per variable when the simulator runs on a computer with a smaller word size. Switching from 'absolute  $\mathbf{x}$ ' to ' $\delta\mathbf{x}$ ' iterations is one of the approaches used in simulators on minicomputers using 32-bit floating point arithmetic [Frer76], [Cohe78], [Cohe81].

Multi-step numerical integration methods require an appropriate time step  $h_{n+1}$  at each point in time. Simulators have a dynamic time step control mechanism which adjusts  $h_{n+1}$  based on two different criteria. The solution  $\mathbf{x}_{n+1}$  obtained with numerical integration differs from the exact solution  $\mathbf{x}(t_{n+1})$  by an amount called the local-truncation error (LTE). This can be written for the  $r$ -th variable  $x_{r,n+1}$

$$\text{LTE}_r = x_r(t_{n+1}) - x_{r,n+1} = C_k x_r^{(k+1)}(\tau) h^{k+1} \quad (2.7)$$

where  $\tau$  is in the time interval  $t_{n-k} < \tau < t_{n+1}$ . The truncation error of the numerical integration method is a useful measure of the correctness of the time step. A more empirical approach is to define  $h_{n+1}$  based on the iteration count of the Newton-Raphson loop at each time point [Nage75]. Both methods are implemented in SPICE2; the local truncation error is estimated based on the charge (or charge components) associated with each element or semiconductor device. The time step is increased when the local truncation error is less than an error bound defined in the program and optionally assignable by the user. The time step is cut whenever convergence fails in a certain number of iterations at a new time point. The iteration time step control proves more conservative but more fool-proof for cases when no charge storage parameters are specified in the model definition of a non-linear device or when the default error bounds are not suited for the example. The approximation of the truncation error is based on an often untrue

assumption that  $x_r$  is continuously differentiable to the order  $k$  in the time interval of interest.

### 2.3. A State-of-the-Art in Simulation

The present trends of the work in the field of electric simulation can be divided in several groups.

- i) Faster simulation for large circuits is of major importance. Some of the reported simulators [Raba79], [Yang80], continue to use direct solution methods together with network tearing and latency to achieve better performance. Another category of simulators developed in recent years [Chaw75], [Fan77], [Newt78], combine equation decoupling, relaxation techniques and selective trace for increased speed. On similar algorithmic basis relies also the Waveform Relaxation technique [Lela82a], [Lela82b], with the major difference that the equation decoupling takes place in time rather than space. The research reported in this dissertation achieves its speed improvement by matching better the circuit characteristics to a vector computer architecture [Vlad81].
- ii) Advances in computer graphics have an important influence on the mode the user communicates with the simulation program. Efforts are presently made to eliminate the encoding of circuit schematic according to a particular syntax which the target simulator accepts and replace it by graphic schematic caption. Real-time graphic output of specified waveforms enables the user to control the simulation, interrupt it and restart it with altered circuit parameters.
- iii) Dedicated hardware simulators [Coh81] which use a shorter word-length on mini- or micro-computer hardware to obtain a performance

and accuracy which is close to those obtained with a medium-size computer.

- iv) Work in device modelling is continued to keep up with the changing characteristics of shrinking devices. Most efforts are in MOSFET modelling and solutions go as far as solving the Poisson, transport and continuity equations on a linear and very coarse grid [Hail82].

The first three directions of new developments suggest a closer interaction between the simulation program and the computer hardware it is run on. The new simulator CLASSIE derives an important part of its speedup from the features of a vector computer architecture. For these reasons in the following two sections both a brief overview of simulation techniques for large circuits and the available computer hardware are presented.

### 2.3.1. Algorithms for Large Circuit Simulation

A fundamental idea included in all new simulators which have been developed in the last few years for large circuits is *Decomposition* [Hach81]. Before giving a classification of the various decomposition techniques it should be emphasized that one class of programs preserves the Newton-Raphson iterative technique and Gaussian elimination, i.e., direct methods only, and achieves the same accuracy as a 'Standard Circuit Simulator' described above. A second group of programs renounces the above algorithms in favor of faster relaxation solution methods based on the Gauss-Seidel (GS) algorithm, e.g., SPLICE [Newt78] and MOTIS-C [Fan77], and Gauss-Jordan (GJ) algorithm, such as MOTIS [Chaw75]. Due to the use of indirect methods these simulators are appropriate for studying certain classes of circuits only. Both above categories are electrical simulators

characterizing the circuits in terms of voltage or current waveforms. While the first category is formed by circuit simulators the second category is referred to as timing simulators.

The decomposition of a large circuit can take place at any level in the solution process shown in Figure 2.1, i.e., at the differential, nonlinear or linear equation solution steps. Two main decomposition techniques are used in the 'third-generation' simulators. The two approaches use different numerical methods with different degrees of accuracy.

#### 2.3.1.1. Tearing Decomposition

Tearing or diakoptics, first introduced by Kron [Kron63], reduces the dimensions of matrices to be inverted by dividing the system into a number of smaller subsystems. Each subsystem is solved as an independent entity and the individual solutions are combined to form the required solution of the total system. In order to interconnect the partial solutions it is necessary to invert an additional *tie (intersection) matrix*.

In electric circuit theory tearing decomposition is an approach which divides a large network into a number of subnetworks which can be processed independently. A number of variables common to more than one subnetwork, called the tearing variables, are solved for in a separate step. The subnetworks (subsystems) can be torn apart across connecting branches or at common nodes. The first approach is called *branch tearing* while the second *node tearing*. In both cases the variables characterizing the torn branches (currents) or torn nodes (voltages) are added to the interconnection (tie) subsystem. A matrix representation of the tearing process for a nodal circuit description is presented in Chapter 3, Section

### 3.4.1.

For an MNA simulation program node tearing is the obvious choice. Throughout this dissertation a large circuit is torn at certain nodes which are called the 'external nodes' of the subcircuit. The voltages at these nodes are the tearing variables and are solved for simultaneously for all subcircuits. The 'internal voltages' of each subcircuit can be found independently of those of other subcircuits. This solution scheme suggests a two-level analysis.

The information on how the large circuit can be torn is obtained in two ways. A first source is the structured input description which is the natural way of representation for a large circuit. This approach is called *hierarchical tearing*. The only time when this approach cannot be used is when the input description is obtained from a layout extraction program which has 'flattened out' all information relative to hierarchy. Although layout extractors are currently used it is believed that a unified representation of the LSI system will contain the information on structure at any level of representation [Kell82].

In *algorithmic tearing* the large network can be divided in subnetworks automatically using a number of algorithms. One such algorithm for node tearing is proposed in [Sang77]. The tearing can be performed on a 'dependency or image' matrix which is a graph (topology) representation of the large circuit. No reported simulator uses algorithmic tearing so far. The major drawback of this approach is that it can not recognize the existence of repetitive patterns (or subcircuits) which can be exploited in the analysis. This shortcoming can be viewed in a similar way as that of a layout extractor which tries to reconstruct the information about transistors and gates from

a huge number of geometrical shapes (rectangles). In both cases, for the purpose LSI/VLSI circuits, the hierarchy information is available at the design stage and should be retained in the circuit representation. For a layout extractor, only the parasitics are found and added to each component of the hierarchy, cell, building block or system.

Decomposition can take place at different stages of the analysis (Figure 2.1). Only tearing (direct methods) is considered in this section. A dynamic system decomposition can be performed at the differential equation level, before discretization. Each subcircuit is solved on a different time scale. This approach takes advantage of the inactivity of a large part of the system at any given time. In logic simulation a scheduling or selective trace algorithm is used in this situation. This technique is however not useful in a circuit simulator (with direct solution methods only) because of feedback paths which must be solved at the same time points as the rest of the circuitry interacted with. Feedback also invalidates the unidirectionality of the signal flow assumed in logic and timing simulators. MACRO, a program developed at IBM [Raba79] reports the use of hierarchical tearing at the dynamic system level.

Decomposition at the nonlinear system level can be achieved in two ways, as implemented in Programs SLATE from the University of Illinois, and MACRO. The first is reported to use modified Newton-Raphson iterations to linearize the equations representing the different subcircuits and their interconnections. The linear equations are solved independently for each subcircuit. MACRO implements a 'multilevel' (actually two-level) Newton algorithm which iterates on the nonlinear equations representing the subcircuits independently from those representing the system. Each subcircuit

is described by its own set of nonlinear algebraic equations of the form

$$H(\mathbf{u}, \mathbf{x}, \mathbf{y}) = 0 \quad (2.8)$$

where  $\mathbf{u}$  and  $\mathbf{y}$  are two different variable representations, e.g., voltage and current, which characterize the torn nodes;  $\mathbf{x}$  are the internal variables. Equation 2.8 is an implicit map from  $\mathbf{y}$  to  $\mathbf{u}$  if  $\mathbf{u}$  are the input variables, known from the system level, and  $\mathbf{y}$  the output variables. The equations at the system level are then

$$F(\mathbf{u}, G_y(\mathbf{u}), \mathbf{w}) = 0 \quad (2.9)$$

where  $G_y(\mathbf{u})$  represents the 'exact macromodel' of the subcircuit, and  $\mathbf{w}$  are the variables at the system level which do not interact with the subcircuits. Since the NR algorithm has a quadratic rate of convergence, the above simulation approach has a quadratic rate of convergence only if

$$\|\Delta \mathbf{x}, \Delta \mathbf{y}\| \leq \|\Delta \mathbf{u}, \Delta \mathbf{w}\|^2 \quad (2.10)$$

where  $\Delta \mathbf{x}$ ,  $\Delta \mathbf{y}$ ,  $\Delta \mathbf{u}$ ,  $\Delta \mathbf{w}$  are the tolerances at the subcircuit and system level, respectively, for which the iterative process is terminated [Raba79].

The circuit can be torn at the nonlinear equation level in yet another way if each transistor is replaced by a submatrix representing the discretized device equations, i.e., Poisson's equation, continuity and transport equation [Laur76].

A few comments are necessary on decomposition by tearing at the differential and nonlinear equation level. A most important aspect is circuit inactivity, commonly referred to as latency. The exploitation of latency is a central issue in simulators for large circuits. As already mentioned it is not possible to implement a scheduling algorithm as is done in logic simulators. In standard circuit simulators, e.g., SPICE2, latency is implemented at the semiconductor device level using a bypass scheme. This scheme is employed

in SPICE2 only at the nonlinear equation level, i.e., if the change in output variables of a device is less than a certain limit at consecutive iterations, at the same time point. The advantage of the bypass scheme is that the Jacobian entries are not computed at this iteration and the values from the previous iteration are retained in the circuit matrix, see Equation 2.5 and Equation 2.6. The problem with this approach concerns how much should a device be allowed to change before it is considered active, e.g., what should be the value of  $\Delta V_D$  for a diode to be reevaluated. The same difficulty is found in a program which defines latency at the subcircuit level as a measure of the change in input/output variables at the torn nodes. This approach might seem straightforward in a program which uses tearing at the nonlinear equation level, e.g., SLATE. Clearly, errors can be avoided in regenerative circuits with long time constants only if all state variables are monitored within the torn block. This technique involves so much computation that latency exploitation loses a lot of its efficiency.

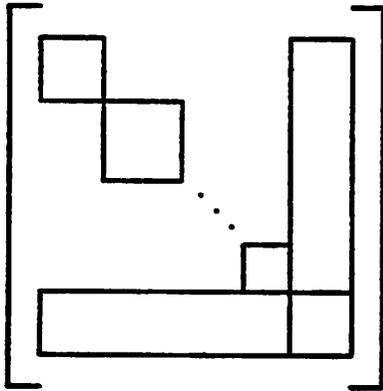
Latency can be exploited efficiently in the tearing decomposition approach only if the network is torn at the differential equation level or, equivalently, in time. In this case, each subnetwork is solved with its own time step which is adjusted based on an estimate of the local truncation error which in turn reflects the circuit activity. MACRO is reported to implement this approach and describes a synchronization mechanism, based on interpolation, to connect the different contributions to the system equations.

Another observation is the two-level analysis implemented by both the MACRO and SLATE Programs using tearing decomposition. With present-day computer architectures the choice of only two levels to perform the circuit

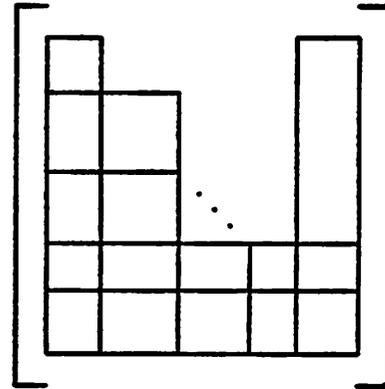
processing, seems the ideal partitioning. Analysis performed at more than two levels incurs an important overhead in connecting the different matrices and increases the computational effort for quadratic convergence in the MACRO approach, see Equation 2.10.

Tearing decomposition can take place at the linear equation level. The different matrix structures which can be obtained by tearing are presented in Figure 2.2. These structures are used by the above programs and by CLASSIE. In electric simulation it is not advantageous to use decomposition after both time dependencies and nonlinearities have been removed. The high sparsity of the matrix, over 95% for a medium circuit, and the use of Gaussian elimination make the equivalent of a matrix inversion to be performed efficiently and preclude the usefulness of tearing. For large circuits described by matrices with a few thousand rows and columns the solution of linear equations can become quite important if a Fortran encoded routine is used. The machine code solvers employed in the two programs SPICEV and CLASSIE, evaluated later in this text, use only up to 10-20% of the total time for the linear-equation solution.

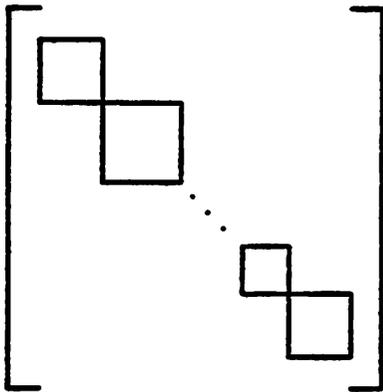
A reordering strategy can be found for the linear equations described by Equation 2.6 which casts matrix  $A$  in a block structure. The four block structures are labelled in Figure 2.2 as the Bordered-Block Diagonal (BBD), the Bordered-Block Triangular (BBT), the Block Diagonal (BD), and the Block Triangular (BT). Tearing applied at the nonlinear and differential equation level generate a BBD matrix. The subcircuit internal variables form the diagonal blocks, the external variables form the border and contribute to the lower right corner; this latter submatrix contains mainly the variables at the system level.



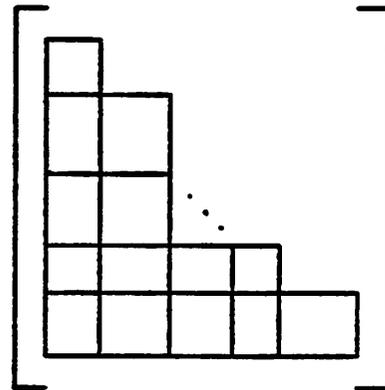
**Bordered-Block-Diagonal**



**Bordered-Block-Triangular**



**Block-Diagonal**



**Block-Triangular**

**Figure 2.2 Triangular Matrix Structures**

### 2.3.1.2. Relaxation Decomposition

The relaxation or temporal decomposition technique stems from the partitioning of the circuit matrix

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} \quad (2.11)$$

into the sum of a strictly lower  $\mathbf{L}$  (diagonal is zero), strictly upper  $\mathbf{U}$ , and diagonal matrix  $\mathbf{D}$ . For a network with  $k$  variables, the solution is found independently of the other variables (GJ) or as a function of the variables already solved for GS. A fundamental assumption necessary for this approach to be sufficiently accurate in the case of electrical simulation is the unidirectionality of the signal flow. In terms of Equation 2.6 the GR and the GS algorithms can be expressed as

$$\text{(GJ)} \quad \mathbf{D}\mathbf{x}^{m+1} = -(\mathbf{L} + \mathbf{U})\mathbf{x}^m + \mathbf{b} \quad (2.12)$$

$$\text{(GS)} \quad (\mathbf{L} + \mathbf{D})\mathbf{x}^{m+1} = -\mathbf{U}\mathbf{x}^m + \mathbf{b} \quad (2.13)$$

A simulator which uses the above equations and equivalently circuit decoupling belongs in the category of Timing Simulators. In the standard-timing-analysis approach the decoupling takes place at the level of each variable. This is called a point-wise decomposition and is used in the timing simulators MOTIS [Chaw75], and MOTIS-C [Fan77]. These simulators have in common the time discretization which is the same for all variables in the system. Another characteristic common to both simulators is the one-shot relaxation performed at each time-point.

The Newton-Raphson has been shown to have a quadratic rate of convergence but is computationally very expensive. More accuracy can be provided in timing simulation if a converged relaxation method is used. One or more Newton-Raphson iterations can be performed point- or block-wise followed by a GS relaxation step. This is the nonlinear SOR (successive

overrelaxation) Newton iteration technique. In a two-level simulator Newton-Raphson iterations can be performed at the subsystem level and a one-step or multi-step relaxation can be used at the system level.

Combining NR with SOR which displays linear convergence is an attempt to get close to the solution first and then use a computationally faster method to converge to the final solution.

If the hierarchical information contained in the input description is used along with the above algorithms, a block-wise relaxation decomposition is achieved. This decomposition technique is used in SPLICE [Newt78], and RELAX which is based on the newer Waveform Relaxation (WR) Algorithm [Lela82a]. [Lela82b]. The advantage of relaxation decomposition is that the system representation can be viewed as a BD matrix, see Figure 2.2, and no time synchronization is necessary as in the case of the MACRO program. Thus, in the RELAX simulator which uses the WR algorithm each subcircuit is solved for the entire time interval independently. The use of the GS relaxation at the system level in RELAX creates the possibility to order the solution sequence according to the signal flow. For each subsystem this ordering makes the fan-in external variables to be the updated solution and the fan-outs to be approximated by the last iteration. Iterations are performed at the system level until all waveforms, for internal and external variables converge within a certain error.

Another characteristic of timing simulators, both point-wise and block-wise, which confers this simulation type an important speed advantage over standard-circuit simulation, is the use of much simplified representations of nonlinear semiconductor devices. In a circuit simulator equivalent conductances are computed corresponding to each controlling variable, whereas in

timing simulation most or all of the dc nonlinearities are transferred to the right-hand side of Equation 2.5. The Jacobian in timing simulation is usually formed by the charge storage elements (capacitors) associated with the devices.

The algorithmic and modelling techniques presented above for timing simulators confer this simulator type a speedup of approximately two orders of magnitude over a standard circuit simulator. The accuracy of mixed-mode (timing/circuit) and WR techniques is considerably better than that of a point-wise, one-shot timing simulation technique due to the iterative nature of the former. The decoupling of the analysis in time allowing optimal time discretization for each subsystem makes SPLICE and RELAX over one order of magnitude faster than a standard circuit simulator, e.g., SPICE2.

### **2.3.2. Computers and Simulation for LSI Circuits**

#### **2.3.2.1. Computer Hardware**

In the last decade much faster computers have been built due primarily to architectural innovation. Advances in semiconductor technology have a secondary role to the advances in computer organization for achieving very high computation rates. Many of the high speed architectures depart from the von Neumann concept of a sequential processing of single instructions fetched from memory based on the program counter. The advent of LSI/VLSI technology has had a decisive role in developing many new architectures which involve more distributed computing. The high density packaging of thousands to hundreds of thousand of transistors in one chip have

increased the reliability of the computer building blocks considerably. The CRAY-1, a state-of-the-art scientific computer, uses only SSI and MSI bipolar Emitter-Coupled-Logic (ECL) circuits for its Central Processor Unit (CPU). Only the recently announced model, the CRAY X-MP [CRAY82], contains VLSI circuits for mass storage, the Solid-state-Storage Device (SSD) replacing the rotating device.

From a design point of view speedup can be achieved two ways:

1. Replicating the instruction and/or data flow constitutes a horizontal speedup;
2. Segmenting (pipelining) the instruction and/or data stream is termed a vertical speedup.

The horizontal speedup architectures can be classified further:

- a. Single Instruction, Single Data (SISD) stream; in this category belong most conventional serial processors outlined by von Neumann in 1949.
- b. Single Instruction, Multiple Data (SIMD) defines a processor performing the same instruction on a set of data; more than one operation can be performed at any one time, i.e., different functional units can work in parallel but under the supervision of a unique control flow.
- c. Multiple Instruction, Multiple Data (MIMD) stream; a number of processors work independently and in parallel on different sets of data towards the solution of the same problem.

Pipelining (vertical speedup) consists in segmenting an operation into a number of segments. This allows more than one instruction to be executed at the same time provided the necessary data paths and functional units are available. The minimum resolvable time defines the clock cycle and a new

instruction can be issued each cycle before the previous instruction(s) have terminated, if the necessary functional units are ready.

A state-of-the-art scientific computer, referred to as a vector or super computer, combines both approaches, horizontal and vertical, to achieve the speeds of hundreds of million of floating-point operations per second. The two commercially available computers which belong in this category are the CRAY and the CYBER 200 computers; both can be classified as pipelined SIMD architectures.

Encoding of scientific problems is often done using arrays. It seems natural to perform an operation between all elements of two vectors (arrays) as one instruction at the computer hardware level. A brief overview of the two processors mentioned above is included here for the understanding of the design guidelines of a program to achieve optimal performance using this computers.

Both processors have an instruction buffer and decode unit, a scalar and a vector processing unit. The memory is organized in banks for fast access of instructions and data each clock cycle. Both have a large number of arithmetic functional units, 13 for the CRAY-1 and 11 or 17, depending on configuration for the CYBER 205. In most of these functional units concurrent processes can take place. In scalar or vector computation a floating-point operation is partitioned into a number of segments and when an intermediate result is ready, it can be chained directly to other functional units [CRAY76], [CDC80]. A resulting vector element is available at each clock cycle on the CRAY-1 and two each cycle on the CYBER 205. The maximum number of segments for an operation is 14 on the CRAY-1 and 26 on the CYBER 205. Beyond these overall similarities there are specifics to

each processor which are outlined in the following.

The CRAY-1, announced in 1975, has been the first of the second-generation vector processors. Its high speed is achieved through ECL bipolar circuitry and carefully dimensioned interconnections. It is characterized by a 12.5 ns clock cycle. Its maximum memory is 4 Mwords of 64 bits each. The processor includes a large number of data registers and address, scalar, floating-point and vector functional units. There are 8 primary- and 64 secondary-address registers (24 bit) which together with the address functional units perform the address calculation and integer arithmetic for the 'short integers'. For scalar processing there are 8 64-bit scalar registers and 64 more secondary registers associated with 7 functional units for arithmetic, logic and shift operations. The vector hardware consists of 8 register files of 64 64-bit elements each and an add, logical, shift and population count functional unit. For vector multiply and divide (inverse approximation) the same functional units are used as for scalar processing. At the Cray Assembly Language (CAL) level, high execution speeds can be obtained by carefully interleaving the instructions for most concurrency in execution and operation chaining.

A new instruction can be initiated at each clock cycle provided no memory, register or functional unit conflict occurs. There is an overhead associated with starting a vector operation but after the result for the first vector element is available each subsequent result follows each clock cycle.

The CYBER 205 implements its instructions through micro-code, separate processing units for scalars and vectors, and uses a smaller cache (256 64-bit registers). This computer has a 20 ns clock and uses virtual memory. The processor is separated into scalar and vector processors. The

scalar processor decodes the instructions, controls the data flow and initiates vector operations in the vector processor. It also contains its own floating-point processor to perform add, multiply, divide, square root and logical operations.

The vector processor contains its own control called a streaming unit which receives decoded instructions from the scalar unit and manages the data stream between memory and floating-point units. There are two floating-point units, called pipes; each pipe has an add, multiply unit and only one pipe has a divide/square-root unit. The floating-point pipes operate on both 32- and 64-bit floating-point numbers. After the start-up time has elapsed each pipe can produce a new result each clock cycle; overall two 32-bit results are computed every clock cycle (20 ns), the pipes working concurrently. For the CYBER 205 there is also a four-pipe option. Both the scalar processor and the vector processor have to share the same 256 registers for intermediate results unless the latter can be redirected to the input of a pipe for a subsequent operation.

The term array processor identifies a single peripheral processor with high-speed floating-point computation capability which can be attached to a general-purpose computer system. The tandem combination usually provides a much higher computation power than the host alone. Although the architectural synopsis and name can cause confusion with the vector computers array processor refers to a distinct category of pipelined SIMD processors. The Floating Point Systems AP-120B and FPS 164 are examples of commercially available array processors. The former is limited by a 38-bit word while the latter is better suited for scientific applications where a 64-bit data word is necessary. The architectural features [Char81] include

multiple functional units, two data register units of 32 registers each, a 1.5 Mword main memory where data and instructions are stored separately, and a 167 ns cycle time. The functional units allow a maximum of two data computations, two memory accesses, an address computation, four data registers accesses, and a conditional branch to be initiated in a given CPU cycle.

Another most important class of high speed processors are the MIMD computers. This architecture is best utilized based on data flow concepts. Data flow architectures allow an instruction to be executed as soon as the operands are available. It minimizes the execution time by assigning its concurrent instructions to separate resources. This systems are in a research stage and do not constitute viable alternatives at the present for the simulation of ICs.

### **2.3.2.2. Vector Processor Performance Evaluation**

A figure of merit is usually needed for comparing the performance of different computers. The standard architectures are characterized in terms of Mips (million instructions per second). A load or an add is used as the instruction time or the average execution time for a mix of instructions. Since vector computers are aimed for scientific computations with a large number of floating-point operations the measure of their speed performance is usually taken as the number of millions of floating-point operations per second (Mflops). The advertised speed of the CRAY-1 is 160 Mflops and that of the CYBER 205 is 800 Mflops. Comparatively the speed of the FPS-164 array processor is characterized by 12 Mflops. These numbers which computer manufacturers promote can be misleading if not put in context. At a first glance the CYBER 205 is five times faster than the CRAY-1.

When large problems are considered with more than one numerical method involved and complex data structures, e.g., circuit simulation, it is not obvious which of the above processors is superior. A program will achieve a larger or smaller percentage of the maximum execution speed depending on the extent it is adapted to the specifics of the architecture and instruction set.

For an understanding of program design considerations and results presented in chapters to follow it is useful to analyze the performance of the two vector processors. The following simple equation, [Cala79], describes the time needed for the execution of a vector operation  $T_v$  :

$$T_v = T_s + \tau T_o \quad (2.14)$$

where  $T_s$  is the start-up time,  $\tau$  the average vector length, and  $T_o$  the time each result is ready. The quoted Mflops rates are derived on several assumptions. First, it is considered that the vector length is infinite and the start-up time is negligible. The speed for one floating-point operation also called a diadic operation is  $\frac{1}{12.5 \text{ ns}} = 80\text{Mflops}$  for the CRAY-1 and  $2 \times \frac{1}{20 \text{ ns}} = 100\text{Mflops}$  for one pipe for the CYBER 205.

Second, all possible concurrency is assumed. Thus, on the CRAY-1 a vector add and multiply, a triadic operation, can run in parallel raising the Mflops rate to 160. On the CYBER 205 a vector add and multiply can run concurrently in each of the four pipes bringing this rate to 800 Mflops.

However, details are missing in these numbers, i.e., the CRAY-1 operates on 64-bit floating numbers only whereas the CYBER 205 operates both on 32 and 64-bit numbers. The above data refer to the maximum speed, i.e., 64-bit arithmetic for the CRAY-1 and 32-bit for the CYBER 205. When both work

with 64-bit numbers the CYBER 205 achieves a maximum speed of 400 Mflops with 4 pipes and 200 Mflops with 2 pipes.

Part of the ideality of the study is removed when the actual vector length is considered. The importance of the start-up time can be recognized when defining the vectorization efficiency  $\eta$  :

$$\eta = \frac{\text{operation time}}{\text{start-up time} + \text{operation time}} = \frac{1}{1 + \frac{1}{r \frac{T_o}{T_s}}} \quad (2.15)$$

A universal efficiency curve based on the above equation is shown in Figure 2.3 [Cala79]. For solving a certain problem it is important to evaluate how much of the raw processor speed can be obtained for typical vector lengths. A first observation based on just the add or multiply operation is that the CRAY-1 has a shorter start-up time than the CYBER 205 [Kasc79]. Thus the CRAY-1 achieves half of the maximum speed for an average vector length

$$r_{1/2\text{CRAY-1}} \approx 15$$

as compared to

$$r_{1/2\text{CYBER205}} \approx 50$$

This assumes that the data are stored contiguously in memory. For short vector lengths (a few tens of elements) the CRAY-1 is more efficient whereas for vectors with thousands of elements the CYBER 205 can be faster. The CRAY-1 has actual 64-element long hardware vector registers and therefore uses an additional overhead for storing and reloading these registers for a computation which involves vectors longer than 64. On the CYBER 205 the longer the vectors it operates on the more the operation time absorbs the start-up time and produces a higher execution rate.

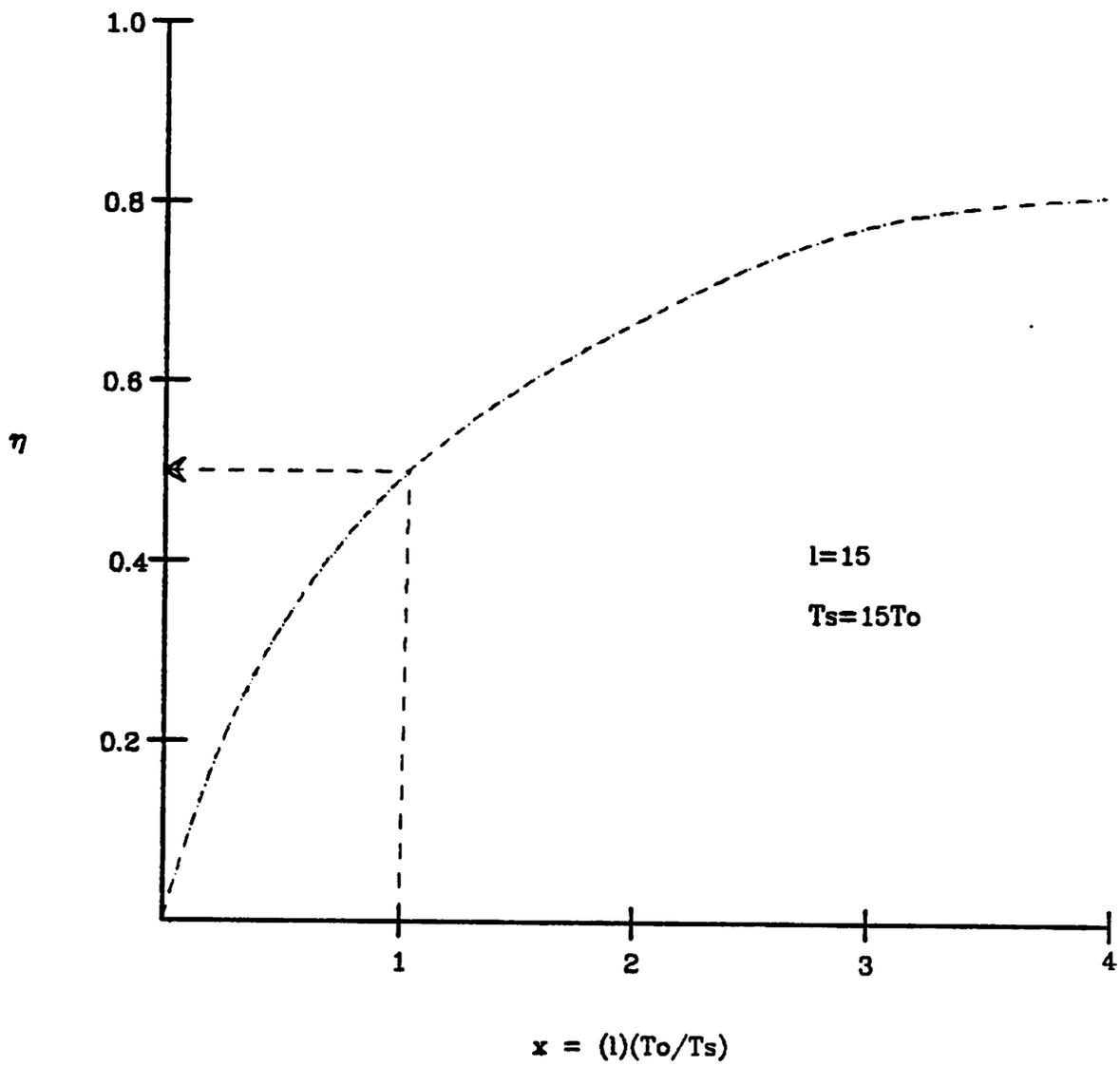


Figure 2.3. Universal Efficiency Curve for Vector Computation

For a scientific application program the most interesting performance measure is an equivalent Mflops rate which incorporates the memory traffic present in any algorithm implementation. In a circuit simulator the only computation section which can be isolated easily is the linear equation solution. The sparse solver is characterized in terms of equivalent Mflops in Chapter 5.

### 2.3.2.3. Algorithm Evaluation on Vector Processors

With the architecture of the vector processor known, the remaining question to be addressed is how to take advantage of the hardware using a high-level language such as Fortran. An obvious correspondence to a vector add is illustrated by the following DO loop:

```
      DO 100 I=1,N
          A(I)=B(I)+C(I)
100   CONTINUE
```

Assume that the elements of the arrays are stored in contiguous memory locations. On a scalar computer the above loop is executed as a sequence

**Load, Load, Add, Store, Index and Branch**

which is repeated until the loop variable has assumed all values. For a large value of N thousands of instructions have to be executed for completion of the loop. This explains why in the context of vector processors it is not realistic to measure the effective speed in Mips.

The same loop will however not be translated as one vector instruction if the elements of array B are pointed to by another array INDEX:

```
      DO 100 I=1,N  
          A(I)=B(INDEX(I))+C(I)  
100    CONTINUE
```

This very simple example demonstrates the importance of data structures; all data which are processed together should be stored in contiguous locations if a program is to take full advantage of vector computation. Data which are expected to be processed with vector instructions can be equally retrieved from memory if stored a constant stride apart on the CRAY-1. In the latter case caution must be exercised not to generate memory bank conflicts.

The operation of loading elements from random locations in memory into a contiguous array is referred to as *gather* operation. The storage of contiguous elements of an array in random locations in memory is called a *scatter* operation. If the data structure does not provide for contiguous elements gather and scatter operations are necessary. These operations contribute a larger or smaller overhead depending on the computer system. For the CYBER 205 gather/scatter operations are a trivial matter because of the existence of a vector gather/scatter instruction. This instruction executes at the rate of one operand every 1.25 clock cycle. On the CRAY-1, however, the gather/scatter operations must be performed by an assembly-coded library routine which needs 20 clock cycles per operand. This time can be reduced to 12 clock cycles if machine instructions are generated and executed for every gather/scatter operation.

Another problem for vectorization of a DO loop is created by the processing of conditional statements or branches in that loop. Assume that there are two analytic formulations for a function depending on the run-time values of certain variables. It is further assumed that both the values

of the function and the test variable form vectors. If a test was to be performed for each element of the variable vector only scalar operations could be used. The approach followed for vector operations is to evaluate both analytic formulations for as many elements as the function and variable vector contain and use two temporary vectors for storage. The values of the elements of the function vector are the result of a merge operation between the elements of the two temporary vectors. Each element of the result vector is picked from the first or second temporary vector depending on the corresponding value in the variable vector. This approach involves more computation than the scalar processing of a conditional loop but overall it is usually faster. A more detailed example based on semiconductor model evaluation is commented in the following chapter.

Additional restrictions apply when a high-level language compiler such as Fortran is desired to generate vector code. Thus, in a DO loop the arguments of the arrays can be only linear functions of the loop variables for vectorization. This type of variable is called constant increment integers (CII) in CRAY Fortran language. Other examples of non-vectorization are dependencies between vector elements used in the arithmetic. Recursion is a typical example:

```

      DO 100 I=2,N
          A(I)=A(I-1)+B(I)
100    CONTINUE

```

A final example follows which demonstrates the difference in speed one can get from a vector processor by using the knowledge of the architecture. The multiplication of two full matrices **A** and **B** is analyzed. Usually element  $c_{jk}$  of the product matrix **C** is

$$c_{jk} = \sum_{s=1}^N a_{js} b_{sk}$$

which is called the inner or dot product of the  $j$ -th row of  $A$  and the  $k$ -th column of  $B$ . The time required by this algorithm is  $N^3 + O(N^2)$  where  $N$  is the rank of the matrices.

The same result can be obtained by a reordering of operations:

$$\begin{pmatrix} c_{1k} \\ \vdots \\ c_{Nk} \end{pmatrix} = b_{1k} \begin{pmatrix} a_{11} \\ \vdots \\ a_{N1} \end{pmatrix} + b_{2k} \begin{pmatrix} a_{12} \\ \vdots \\ a_{N2} \end{pmatrix} + \cdots + b_{Nk} \begin{pmatrix} a_{1N} \\ \vdots \\ a_{NN} \end{pmatrix}$$

The product of two matrices performed using the above sequence of operations is called the outer product. The equivalent Fortran is:

```

DO 200 K=1,N
  C(1,K;N)=B(1,K)*A(1,1;N)
  DO 100 L=2,N
    C(1,K;N)=C(1,K;N)+B(L,K)*A(1,L;N)
  100 CONTINUE
200 CONTINUE

```

The notation  $C(1,K;N)$  represents all  $N$  elements (a vector) of the  $K$ -th column of matrix  $C$ . The vector statement in the inner DO loop executes an add and a multiply concurrently and is called a triad. Each result as it is available from the multiply unit is chained to the input of the add unit. The execution time for this operation is thus reduced by a factor of two on any of the vector processors or array processors introduced earlier because of the parallel processing of the two floating-point operations in independent functional units. An additional factor of two speed improvement can be obtained on a CYBER 205 with four pipes.

A few concluding remarks on vector computation maybe helpful. The speedup achieved from a vector or array processor depends on how well the

algorithm and the data structure make use of the architecture. The high-level language compiler limits the possibility of taking full advantage of the architecture and ratios of 5:1 in speed between hand-coded assembler and Fortran produced code have been reported [Cala79]. The constraints of the Fortran compiler on the programmer force the latter to understand the data flow. Researchers at the University of Illinois, have written a Fortran compiler, Paraphrase, [Wolf78], which performs a careful data dependency and data flow analysis for parallel processing. This approach helps the programmer by removing several constraints of commercially available compilers but does not free him from carefully designing the algorithm and data structure.

No existing electrical simulation program has been designed or redesigned to use vector processing. They can be used however on vector processors and obtain good performance due to the raw speed increase of these new computers. The speedup compared to a standard scalar computer, e.g., the VAX 11/780, varies depending on the program. Thus the speedup on the CRAY-1 for SPICE2 is twice the speedup for MOTIS [Chaw75]. This can be explained by the fact that the CRAY-1 performs much more efficiently than a scalar processor the floating-point operations, exponentiations, logarithms, and square roots, which are abundant in SPICE2 compared to the memory operations and less computation which is typical for MOTIS.

#### **2.4. Semiconductor Device Modelling**

The computation of the partial derivatives which constitute the elements of the Jacobian is done on the nonlinear functions describing the I-V, Q-V, and/or C-V characteristics of the semiconductor device. Although

other nonlinear elements can be included in a circuit simulator, the usual ones stem from the nonlinearities of the active devices. The process of finding the elements of the Jacobian for semiconductor devices is both very important for the accuracy of a simulation and the run time.

In existing simulators two basic approaches are used for describing the nonlinear behaviour of active devices. Some programs, such as SPICE2, have built-in models for the most common semiconductor devices, diodes, Bipolar Junction Transistors (BJT), Junction Field-Effect Transistors (JFET), and Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFET). Other programs, viz., ASTAP, allow the user to describe the nonlinear characteristics as analytical functions as part of the circuit input specification. Although the second approach seems to offer more flexibility it is made impractical due to the constraints which the analytical description must fulfill in order to preserve the convergence of the iterative solution. Often the user must resign and use a simple model which is found in the model library and works.

The constraints which need be respected when developing a new model for circuit simulation include the following:

- i) The analytical model functions should be continuous and strictly monotonic with continuous derivatives over the whole range of device operation.
- ii) The model definition is desirable to be a simple and explicit function of independent variables, e.g., terminal voltages.
- iii) It is important to choose a best suited set of independent variables for an uniform and computationally efficient representation; examples in this sense are transport vs. injection models for the bipolar transistor [Getr76] and source-referenced ( $V_{GS}$ ,  $V_{DS}$ ,  $V_{BS}$ ) vs. bulk-referenced

( $V_{GB}$ ,  $V_{DB}$ ,  $V_{SB}$ ) models for the MOSFET.

The first requirement is important for convergence purposes while the second relates to computational efficiency (avoids nested iterative loops). The last requirement is to provide effective ways of extracting parameters for the new implemented model.

A common approach is to derive a set of I-V and Q-V equations which satisfy the above conditions. In SPICE2 the computation of all conductances (partial derivatives) is based on analytical formulations. This approach makes the implementation of a new model a lengthy process and can introduce discontinuities in the partial derivative formulation. A faster approach is to program just the I-V and Q-V equation and to compute the values of the Jacobian entries using numerical differentiation. Due to the iterative nature of the solution the above way of derivative evaluation has been considered too approximate. The recent introduction of this method into a commercially available version of SPICE, [Hail82], proves that despite a certain increase in analysis time the accuracy of the derivatives is sufficient to preserve the convergence properties of the program. Another commercially available circuit simulator, ASPEC [Jenk82], replaces the partial derivatives by an equivalent resistance which varies with the trial operating point.

Another method for semiconductor device representation are 'Table Models' [Chaw75]. This type of device description has been first used in MOS timing simulation. More accurate approaches to MOSFET table model implementation in circuit simulation have been proposed [Newt80], [Tana80]. However the highest speedup ever quoted for a table model used in a circuit simulator varies from 2, [Tana80], to 3-4, [Newt77]. The higher figure results partly from a reduction in the number of partial derivatives, only  $G_{DS}$  for DC.

computed as Jacobian entries. There are practical advantages for using table models, i.e., accurate data can be available from measurements, no analytical model derivation and implementation must precede the simulation of a device, no parameter extraction step is necessary. Table models can be implemented by several one-dimensional tables [Newt80], [Tana80], or by two- or three dimensional tables [Shim82]. The storage requirement and search time can become a problem if many different model tables are used. This might be necessary because of the poor scaling properties of I-V characteristics for widely varying channel widths and lengths. In the conclusion of the present project it is shown that for a super-computer with the floating-point capability of the CRAY-1 the savings with table models would be less than 10%.

Faster simulation is obtained using a 'Bypass' scheme. The conductances of the last iteration for a particular device are reloaded if the terminal voltages have not changed. The time-consuming reevaluation of a set of equations is thus avoided. The only danger of a bypass scheme is that for an inappropriate set of tolerances the convergence can be affected. Thus, a device which has changed slightly but not enough to be sensed by the coarser tolerances can be bypassed and reevaluated only in a later iteration. This increases the number of iterations or can cause the solution to diverge. The problems associated with device bypass are similar in nature to those related to the latency of subcircuits described in Section 2.3.1.1.

Two MOSFET models for small-size devices have been implemented in SPICE2. These models have had extensive use for over two years and are described in [Vlad80]. The description of the charge model is included in Appendix 1 for completeness of the SPICE2 MOSFET model documentation.

## 2.5. Convergence

The NR algorithm has quadratic convergence properties assuming that an initial guess is provided which is close to the solution. Therefore it is important in a circuit simulator to provide an initial guess either as a set of node voltages or terminal voltages for the nonlinear semiconductor devices. Circuit simulators start out usually with all unknowns set to zero. SPICE2 initializes the semiconductor devices in a trial operating point such that nonzero conductances can be loaded into the MNA circuit matrix at the first iteration.

The nonlinear I-V characteristics of semiconductor devices are exponential or quadratic functions. With a limited range of floating-point numbers available on a computer and the unbound nature of solutions provided by the NR algorithm according to Equation 2.4 the iterative scheme can fail to converge. A limiting algorithm is needed, as has already been shown in Section 2.2, which selectively accepts the solution unchanged or limits it when large changes in the value of the nonlinear function would occur.

Both the initial trial operating point and the limiting algorithm of the current updating type, [Nage75], works quite well for BJTs. All transistors are slightly turned on in the linear active region.

MOSFET circuits have displayed more convergence problems than the bipolar circuits. The convergence problems can be categorized as failure to find a dc operating point and aborted transient analysis due to the reduction of the time step beyond a certain limit without finding a solution.

Different limiting techniques have been proposed in recent years, [Ho77], [Yang80], but the convergence problems in circuit simulation have

not been completely solved. Ho et. al. suggest a logarithmic limiting of the controlling variable of each nonlinear branch. The logarithmic function is proposed because of the property to leave small changes unaltered and dampen large changes. This approach has been tested in SPICE2 and did not show any improvement. Yang proposes a piece-wise nonlinear approach which separates a nonlinear function into several regions. He outlines a methodology for limiting the change on variables predicted by the NR iteration when this change crosses boundaries between regions. A generic load resistor 'R' is used for defining the boundary between different regions. The idea of the nonlinear characteristic breakpoints is not new and has been used in circuit simulators such as SPICE, BIAS3, etc., for the limiting routines. The author also falls short of specifying a way of computing the generic load resistor 'R' for a random transistor in the circuit.

There are a number of approaches which can improve the convergence of the circuit solution. Most of the comments presented in the next paragraphs are based on the SPICE2 simulator. The initial operating point for semiconductor devices has been found to be very important. For bipolar circuits which are slightly turned on in the initialization phase a smaller number of iterations have been noticed for analog (linear) circuits as compared to digital (logic) circuits. The explanation can be found in the mode of operation of analog circuits which have the majority of the transistors turned on in contrast with digital circuits which have an important percentage of the devices turned off. MOS circuits have the transistors turned off in the first iteration. Initializing all MOSFETs in the conduction state has been found to speed up the convergence of operational amplifiers and other analog circuits. These observations suggest a user option for the state of transistors in the first iteration. An additional note concerns the

connection of the transistor terminals; the initial operating point should take into account if two or more terminals are tied together, e.g., a depletion or saturated load, a diode connected BJT, etc., and set the terminal variables accordingly.

A convergence improvement has been obtained in SPICE2 for MOSFETs by changing the variables on which limiting is performed. These terminal voltages used to be  $V_{GS}$  and  $V_{GD}$ . The controlling voltages for a MOSFET transistor are actually  $V_{GS}$  and  $V_{DS}$  which are at the same time the independent variables in the device equations. The limiting is different for these two variables; in the  $I_{DS}-V_{DS}$  plane the breakpoints are  $V_{DS} = 0$  and  $V_{DS} = V_{DSsat}$ , while in the  $I_{DS} - V_{GS}$  plane regions of operation are separated at  $V_{GS} = V_T$ . A number of analog MOSFET circuits characterized by high impedance nodes, e.g., current sources, which failed to converge on previous versions can be successfully simulated.

Two other approaches are used effectively for finding the dc solution. Both ASPEC [Jenk82] and ASTAP [Week73] improve the dc convergence by solving actually a one-step transient analysis of a network having a capacitor connected at each node to ground. This approach creates a diagonal dominance and reduces the importance of the state in which the semiconductor devices are initialized. The other method for improving dc convergence is known as source stepping. All independent voltage and current sources are initially zero values which are consistent with an all-zero initial solution vector. The dc operating point is found by ramping all sources up to the actual value, similar to computing a transfer characteristic. This approach can be implemented as a back-up and used only if convergence has failed in a given number of iterations.

## 2.6. Linear Equation Reordering and Solution

Nodal analysis generates an admittance matrix which is very sparse and usually has diagonal dominance. Introducing current equations in the MNA formulation affects the diagonal dominance. There are three important issues which are commented upon in this section related to the reordering of equations for accuracy, the sparse matrix pointer system and the vectorization features of sparse equation solution.

Numerical accuracy problems can be classified as either topological or numerical. The voltage-defined elements, e.g., voltage sources and inductors, generate zero diagonal elements linked to the current equation [Ho75]. This problem can be corrected in the setup phase based on a topological reordering. The row in the MNA matrix corresponding to the current equation is swapped with the '+' node equation [Coh81].

A second problem, which is also topological in nature, is a cutset of voltage-defined elements. This leads to a cancellation of a diagonal element value during the factorization process. Earlier versions of SPICE2, up to version F, which used only topological reordering could not correct this problem. A reordering algorithm has been proposed [Yang80] which finds an equation sequence free of any topology-related problems.

Not all problems associated with the solution of the MNA matrix are topological however. The limited number of digits in the mantissa of a floating-point number (12-13 decimal digits for a 64-bit real) can lead to the loss of significance of a matrix term relative to another during the solution of the linear equations. An LU factorization is used for the linear equation solution. This approach which is similar to Gaussian elimination, decomposes the MNA circuit matrix  $A$  in a lower left  $L$  and an upper right  $U$  matrix:

$$\mathbf{A} = \mathbf{LU}. \quad (2.16)$$

The right-hand side of Equation 2.6 is then modified in the forward substitution step

$$\mathbf{U}\mathbf{x} = \mathbf{b}' = \mathbf{L}^{-1}\mathbf{b}. \quad (2.17)$$

The solution vector is obtained as result of a back substitution

$$\mathbf{x} = \mathbf{U}^{-1}\mathbf{b}' \quad (2.18)$$

The loss of accuracy can be observed in the case of a ring oscillator with high gain stages; the off-diagonal elements are transconductances which grow as a power law of the gain factors during factorization and eventually swamp out the diagonal term.

Another example of a numerical problem is a dynamic MOS circuit where some nodes can have a very small conductance for certain clock periods. These examples demonstrate that a topological reordering is not sufficient and that the MNA sparse matrix order has to be based on actual values generated by the circuit. Sometimes it is even necessary to reorder during the iterative process. Reordering based on pivoting is considered vital for large matrices where the original values can be altered significantly by the factorization process. A more detailed description of the above circuit examples can be found in [Cohe81].

In SPICE2 the topological aspects are considered in the setup phase when the sparse-matrix pointers are defined. The numerical reordering, based on partial pivoting and the Markowitz algorithm for minimum fill-in, is performed at the very first iteration after the actual MNA values have been loaded. Fill-ins are the matrix terms which are zero at the beginning of the factorization process and become nonzero during the execution of the LU decomposition. Pivoting is performed on the diagonal elements; only if no

pivot can be found on the diagonal the rest of the submatrix is searched. Once an optimal order has been found it is used throughout the analysis unless at any point a diagonal element does not satisfy an absolute minimum criterion. In this case 'pivoting on the fly' is performed for the remainder of the matrix. In the pivoting phase a relative error bound is used for comparison with the maximum column entry.

The choice of a sparse matrix data structure must consider the need for a change of order and the addition of new terms (fill-ins) at any point during the analysis. For this purpose a set of four bidirectional linked lists are used in SPICE2. The actual matrix values are stored in an one-dimensional array of reals. The information on a certain element as to storage location, actual position in a two-dimensional matrix and value can be retrieved using the same offset in the tables pointed to by the sparse-matrix descriptors. This methodology is due to [McCa76] and has been used successfully in program MICE [Cohe78] before it was implemented into SPICE2 [Vlad78].

An additional set of pointers stores the correspondence between the internal circuit node numbers and the equation numbers. Independent row and column swap can be performed. When retrieving the solution which is stored in the same locations as the RHS, two mapping operations are required to compare it with the previous solution which is stored by node numbers. Because this can be a lengthy process for large systems an additional table can be stored which provides the direct mapping between the solution vector and the circuit nodes.

Vectorization of the sparse linear equation solution is an important issue for large circuits. For special types of sparse matrices, such as

banded matrices, efficient solution algorithms have been developed which use vector arithmetic. The sparsity pattern of MNA matrices describing ICs, however, is very random. Calahan, [Cala79], reports an attempt to vectorize the solution by defining line vectors on the CRAY-1. The average vector length found in the above work during the factorization of an  $870 \times 870$  matrix is 1.99! In this case it is more efficient to use scalar operations.

## CHAPTER 3

### CLASSIE - Design Considerations

#### 3.1. Introduction

SPICE2 has become a standard in circuit simulation throughout industry due to the robustness of its algorithms, reliable implementation, portability and continued work to keep it up-to-date with IC technology. There are however two major factors which point to the fact that a faster and more efficient simulator can be built to perform an equally accurate circuit-level analysis. The two factors are the characteristics of an LSI circuit and the state-of-the-art commercial computer system, the vector computer.

In this chapter the performance of SPICE2 is examined on different computers and conclusions for the design of the new program, CLASSIE, are drawn. The design decisions presented in this chapter start with the specifics of LSI circuits which suggest a hierarchical analysis, performed at two levels, the cell (subcircuit) and system (interconnection) level.

The requirements of the two-level analysis is reflected in the choices made for the most important components of the analysis. The linear equations of the LSI form a Bordered-Block-Diagonal Form (BBDF) matrix which is solved by machine code generated by the program. The semiconductor devices are sorted according to subcircuit or model parameters and are linearized in vector mode. The data structure is built around individual tables for each element. The two-level analysis requires unique integer

tables for all instances of a certain cell type and real tables with multiple copies of the values for each cell instance. The entries for different cells are aligned for efficient vector operations.

### **3.2. Circuit Simulator Performance and Computer Hardware**

Initially developed on the CDC 6400 campus computer of the University of California at Berkeley, SPICE2 has been modified to run on other computers by interested parties. In 1979 an easy transportable version (SPICE 2F.1) has been introduced to replace the older CDC oriented-version. From a single source code with embedded conditional compile control statements six different compile codes are generated for CDC, IBM, VAX/UNIX, VAX/VMS, CRAY, and HP3000 computers. There are a few differences between the two SPICE2 source codes for the VAX 11/780 depending on the operating system which is used, UNIX or VMS. Because of the availability of the above code it became possible to compare the performance of the same computationally involved simulator on different hardware, architectures, Fortran compilers, and memory systems.

In the design of CLASSIE the primary goal has been to match the specific features of LSI circuits and computer architectures as closely as possible. The expected result is a faster simulator than SPICE2 when both programs run on the same hardware. Besides the improved performance due to the CRAY-1, CLASSIE exceeds the speed of SPICE2 on other non-vector machines as well due to general features such as more locality in the data structure, an advantage on virtual memory machines, and organization of data by categories, which reduces search time. The recognition of parallelism in circuit simulation creates also the basis for a future multi-

processor environment.

The SPICE2 program uses a linked-list data structure to store the circuit description in computer memory. The data storage and memory management scheme of SPICE2 are appropriate if the circuits are reasonably small, a few tens of elements, and memory access is unimportant compared to the execution time of a floating-point operation on the host computer. None of the above assumptions hold true any more today. It is common that circuits analyzed in industry have one hundred and more components; computers have floating-point accelerator units at the lower end of the scale and multiple arithmetic units at the high end. Most memory systems use a virtual addressing scheme to page in and out the data which the program needs at each step.

There are two other major aspects of the host computer which decide the performance of an application program such as SPICE or CLASSIE. The first is the architecture, the number of internal registers, existence of a cache or not, etc. Second the high-level language compiler has a decisive role on how well the architecture resources are used. Only about one fifth of the performance of a very complex architecture can be achieved through the Fortran compiler in the case of a matrix factorization [Cala79]. The examples in this section illustrate for computers having a rather conventional architecture, the VAX 11/780, how the compiler has an important impact on the circuit simulator performance.

The conclusions reached from the careful analysis of the run times on different benchmarks and their breakdown among different sections of the program led to the design decisions for the new program. The run statistics of three circuits on four different computers are presented in this section.

	Circuit	Type	Devices	Eqs	Time Pnts	Iter
1	UA741	Bipolar	22	52	67	187
2	LOWPAS	MOS	70	48	165	450
3	ADDER4	Bipolar	288	450	4653	25171

**Table 3.1. Transient Analysis Statistics of Three Benchmarks**

The three selected circuits are of small to medium complexity, from a few tens to a few hundred semiconductor devices. The UA741 operational amplifier has been a long time SPICE benchmark [Nage75], [Cohe81], and can be categorized as a small bipolar circuit. A low-pass section of a switched capacitor filter is a second small example with its 70 MOS transistors. The medium-size example is a 4-bit adder built with bipolar NAND gates. The SPICE2 input files for the above two bipolar circuits are listed in Appendix 2.

Four computers which run the same version of SPICE2 are used in this comparison. An older scientific computer, the CDC 6400 which is not included in this comparison, provides run times comparable to the newer VAX 11/780 computer. At the high end the CYBER 175 offers a measure for one of the fastest scalar (Single-Instruction Single-Data) computers and the CRAY-1 represents a state-of-the-art vector (Single-Instruction Multiple-Data) computer. The two data entries for the VAX are measured on machines running UNIX or VMS operating systems with different Fortran compilers.

Table 3.1 gives a summary of the three benchmarks and the run statistics thereof. Table 3.2 contains the central processor times per iteration for the transient analysis of the circuits enumerated above. A normalized speed for a SPICE2 run is computed based on the CP times of the different computers in comparison to the CP time for the same run on a VAX/VMS.

Conclusions to be drawn from the above tables include an inherent speedup that comes with the computation capability of the hardware. Thus there is a factor of 30 between the run time on the CRAY-1 and on a VAX/VMS computer. The latter has been taken as a reference because of its

CPU	CP/ITER (ms)			NORM. SPEED		
	UA741	LOWPAS	ADDER4	UA741	LOWPAS	ADDER4
CRAY-1	3.82	12.31	30.56	26.18	30.82	23.84
CYBER 175	14.41	47.65	118.86	6.94	7.96	6.13
IBM 4341	52.42	213.76	367.2	1.91	1.78	1.98
VAX/VMS	100	379.42	728.55	1	1	1
VAX/UNIX	184.38	871.93	1,278	0.54	0.44	0.57

**Table 3.2. SPICE2 Simulation Speed as a Function of Host Computer**

wide-spread use in the semiconductor industry. On the same hardware, such as the VAX 11/780, the Fortran compiler can have an impressive effect on the performance of the program. The code produced by the VMS Fortran IV compiler for SPICE2 runs approximately twice as fast as the code produced by the UNIX Fortran77 compiler. The timing information for the Cyber 175 is obtained on the medium speed memory version (Cyber 175/200). On the high-speed version, SPICE2 is only 2.5 times slower than on the CRAY-1 as compared to 4 in Table 3.2. The CDC 6400 imposes serious limitations on the circuit size because of its reduced data space (40k words) for SPICE2. Only the transient response for the UA741 could be run in this amount of memory. Beyond the speed ratio there are other differences in the data representation (floating-point numbers in particular), floating exception interrupt handling, etc., which impact the results of the same run on a different hardware and/or operating system. There are additional features, e.g., code generation for the solution of the linear system of equations, which generate additional discrepancies among the different run times.

Table 3.3 shows a breakdown of the analysis time between the two major parts of a circuit simulator: semiconductor-device model evaluation and linear-equation solution. Both the data for the Fortran solver and generated machine instructions (where available) are displayed. A first observation is that for the same circuit the relative importance between the two parts differs with the computer. Thus on the CRAY-1, which is a highly pipelined machine with multiple arithmetic units, the percentage of the model evaluation is smaller when compared to the other general-purpose processors. The CRAY-1 is very fast for floating-point computation but is slowed down when significant memory traffic occurs. The linear-equation solution

part (when in Fortran) requires more extensive search (memory references) than floating-point operations. This argument explains both the smaller importance of this part on machines with slower arithmetic and the higher gain for the CRAY-1 when code generation is used.

In the second part of Table 3.3 two numbers are listed for the percentage of the total analysis time used by the machine code solver. The first number represents just the machine code run time while the second number includes also the contribution of the one-time Fortran pivoting reordering, solution and generation of the machine instructions. It can be noticed that the overhead for reordering and machine instruction generation on the CRAY-1 for a small circuit and a short transient analysis, 67 time points and 177 iterations, is approximately four times larger than the run time of the code solver. For medium and large circuits the overhead for generating the machine code becomes unimportant due to the large number of iterations (in the thousands typically). On the other computers the gain of the machine code is far less dramatic. The data on the ADDER4 circuit in the MACHINE CODE SOLUTION column are obtained from transient analyses which have run for over one thousand iterations on all computers but the total number differs from one computer to another. The length of the transient analysis made it necessary to stop it after a given amount of CPU seconds. This may affect the ratio of the actual code execution time to the time which adds to the former the time it takes to reorder the matrix and generate the machine code.

For the simulation of large circuits it is interesting to know how the cpu run time is affected by the increase in circuit complexity. Five bipolar adders from an 1-bit adder to a 16-bit adder is plotted for two different

CPU	MODEL EVALUATION (%)			EQUATION SOLUTION (%)		
	UA741	LOWPAS	ADDER4	UA741	LOWPAS	ADDER4
CRAY-1	42.2	73.4	40.7	42.2	11.9	57.2
CYBER 175	44.4	76.7	44.5	40.7	11.9	53.6
IBM 4341	58	83.7	57.4	27.6	6.4	41.4
VAX/VMS	53.6	82.35	54.5	30.3	6.37	42.58
VAX/UNIX	62.4	85.1	64.4	20	4.3	33.6

CPU	MODEL EVALUATION (%)			MACHINE CODE SOLUTION (%)		
	UA741	LOWPAS	ADDER4	UA741	LOWPAS	ADDER4
CRAY-1	53.3	79.5	69.6	5.6 (27)	1.7 (4.6)	11 (17)
CYBER 175	57.5	83.5	81.4	5.4 (23.4)	1.7 (3.9)	11.5 (15.3)
IBM 4341	67.6	87	79.5	8.3 (15.7)	2.1 (2.9)	17 (18.1)

Table 3.3. Relative Importance of Model Evaluation vs. Equation Solution

analysis types, two computers and two SPICE2 program versions in Figure 3.1. The difference between Version F and G of SPICE2 is the matrix reordering scheme. In SPICE 2F the row and column order is established in the setup phase based on an image matrix of the nonzero locations. In SPICE 2G the matrix reordering uses pivoting on the actual values loaded at the first iteration. This process is part of the DC operating point analysis, DCOP, and takes place simultaneously with a large number of memory operations. The effect of the reordering is the exponential increase of the cpu time with circuit complexity with a power of 1.3 for the VAX and 1.5 for the CRAY-1. In the absence of pivoting reordering the cpu time increases only with an exponent of 1.05 for larger circuits. This exponent is found from the plots of Figure 3.1 for both the DCOP of SPICE 2F and the transient analysis, TRANAN, of SPICE 2G. The transient analysis of the adders is faster on SPICE 2G compared to SPICE 2F due to less fill-ins introduced by the pivoting reordering. Both programs use the Markowitz criterion for minimum fill-in generation [Chua75].

Several conclusions can be reached on the basis of the data presented so far on SPICE2:

- A circuit simulator picks up a factor of approximately 30-50 on the CRAY-1 compared to common SPICE2 host computers of the super-mini class, e.g., VAX 11/780 or PRIME 850, due to the inherent hardware speed;
- The code produced by the high-level language compiler provides only a fraction of the computer's potential; the statistics show that crucial parts, e.g., the equation solver, of the new simulator have to be coded in assembler.

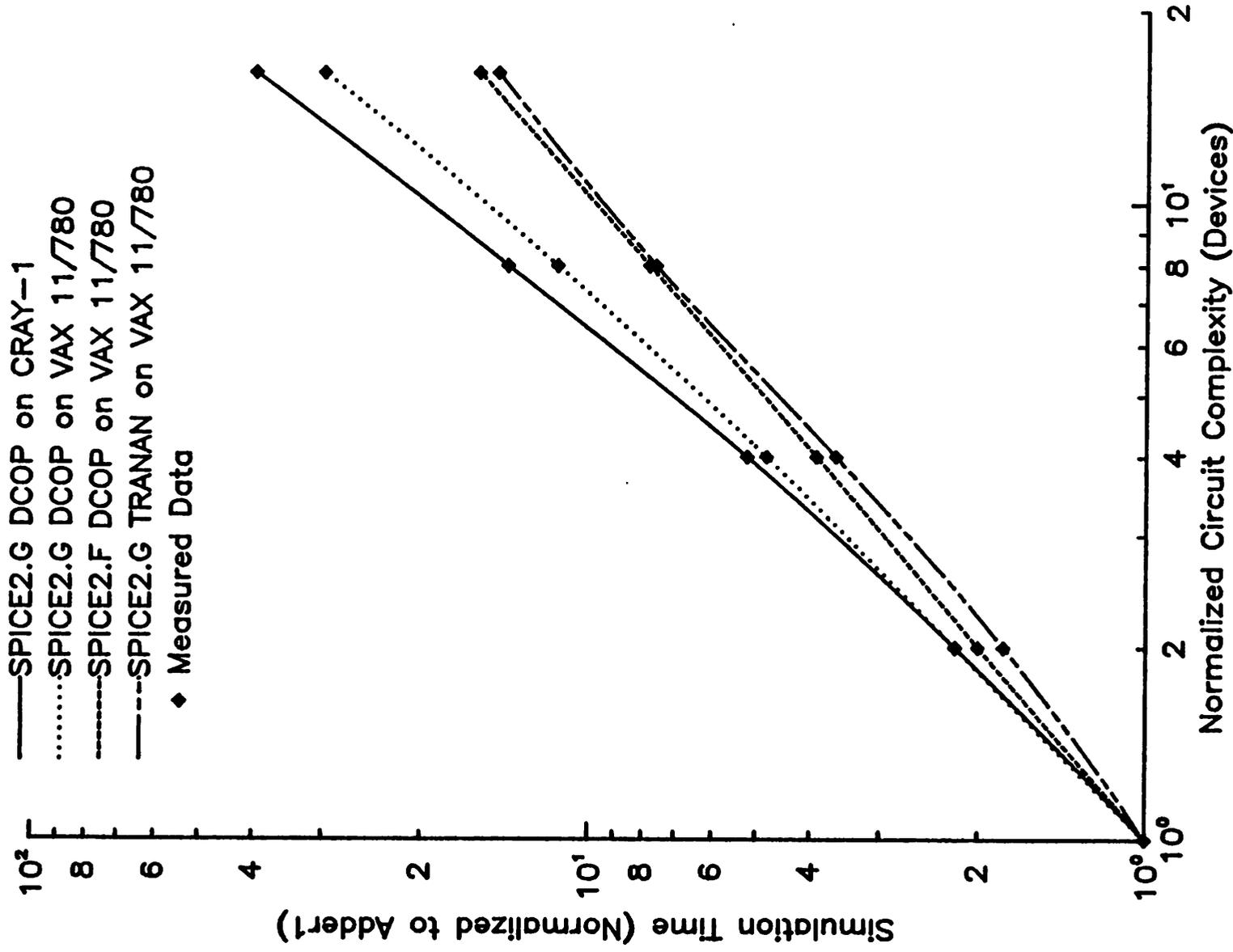


Figure 3.1 Effect of Pivoting Reordering in DCOP Analysis

- At the state-of-the-art floating-point processor speed, the time spent in memory traffic starts to take precedence over the arithmetic operations. The data structure is designed based on these observations to keep the load/store time at a minimum.

### **3.3. Integrated Circuit Specifics and Representation**

One of two basic considerations in the design of the new simulator is the object of the analysis. Only simple circuits had to be analyzed when SPICE was designed ten years ago. These same circuits constitute today mere cells of the LSI circuit. For the purpose of the simulation, an LSI circuit can be described as a collection of a limited number of structurally different functional blocks such as logic gates, operational amplifiers, etc., each block occurring more than once at the system level.

A factor which has to be taken advantage of is the hierarchy (structure) of the circuit. The 'SUBCIRCUIT' feature in SPICE2 is the mechanism to provide the information about structure. However it is a user convenience rather than a source for hierarchical analysis. The partitioning of an LSI/VLSI circuit into a cell / building block / system structure is useful information at any level of simulation. The analysis in the new program is done at the cell (SUBCIRCUIT) level. The structured input description provides the information necessary for an important part of the computation to be performed in parallel. Therefore, the new program groups the identical cells together and performs a two-level analysis [Cala80], [Vlad81a].

An important guideline in the design for speedup is that the number of items which define a vector be maximum. Since CLASSIE gets its hierarchy information from the user-specified input a feature is needed to describe all

the cells with the same topology by the same 'SUBCIRCUIT' definition. Provision has to be made in the input language to allow for some parameters of the devices in the definition to assume values at instantiation only. The instantiation of a subcircuit is the process which adds the collection of elements contained in the definition to the circuit. Figure 3.2 shows an example of the desired parameter passing feature between 'X' call and subcircuit definition. It can be assumed that the definition describes an output buffer; all the instances of this buffer have the same topology but some transistors have different dimensions. The use of the same definition in all cases will force the program to treat all buffers as the elements of one vector and solve them in one pass of the code. The number of instances cannot be increased by reducing indefinitely the circuit size. For a subcircuit with only one internal node the corresponding submatrices reduce to diagonal elements in the overall circuit matrix. A practical minimum requirement for a subcircuit is to contain at least two internal nodes.

Another necessary feature for the convergence of large circuits is the ability to define initial conditions local to each cell instance. The approach used in SPICE2 is to make the nodes to be initialized external. This is particularly detrimental in the setup used in CLASSIE where the internal nodes must exceed the external ones. The explanation for this constraint is given in the following paragraphs.

As has been already shown in the previous chapter the partitioning of the circuit in cells at the lower level and system (or interconnection circuitry) at the upper level, can not use algorithmic tearing because it does not guarantee to identify all the blocks which have the same topology. This would defeat the purpose of exploiting the vector architecture to perform as

## INPUT FORMAT

```
.SUBCKT OUTBUF 1 2 3 4 L2 W2 VDD
.
.
M2 5 2 6 0 CMOS L=L2 W=W2
.
.
.IC V(4) = VDD
.
.
.ENDS OUTBUF
.
.
XOUT1 7 9 4 2 OUTBUF 5U 10U 3
```

Figure 3.2 Parameter Passing between Subcircuit Call and Definition

much computation in parallel as possible.

### 3.4. Algorithms and Implementation

#### 3.4.1. Modified Nodal Admittance Matrix

SPICE2 operates on an entire circuit matrix which is loaded element by element. The analysis in the new program is done at the functional block (subcircuit) level. However, only direct methods are used in the solution procedure, i.e. no decoupling takes place at any level as in timing or mixed-mode simulation.

Each functional block (subcircuit) generates a diagonal submatrix in the overall circuit matrix. Thus the circuit can be represented as a Bordered Block Diagonal Form (BBDF) matrix as shown in Figure 3.3. At the lower level there are the diagonal submatrices representing the internal nodes of each subcircuit and at the upper level there is the interconnection matrix (borders and lower-right corner). The linearization and solution of subcircuits can be done independently of each other and the interconnection, and can be done simultaneously in the vector-mode for those of the same type. This is possible because all these subcircuits have the same topology. The feature of parameter passing between subcircuit call and subcircuit definition increases the occurrence of topologically identical subcircuits.

The solution proceeds as follows. All subcircuits are represented by the following set of equations:

$$\begin{bmatrix} \mathbf{Y}_{ss} & \mathbf{Y}_{si} \\ \mathbf{Y}_{is} & \mathbf{Y}_{ii} \end{bmatrix} \times \begin{bmatrix} \mathbf{V}_s \\ \mathbf{V}_i \end{bmatrix} = \begin{bmatrix} \mathbf{C}_s \\ \mathbf{C}_i \end{bmatrix} \quad (3.1)$$

For the purpose of simplicity it is assumed that the Y matrices are pure

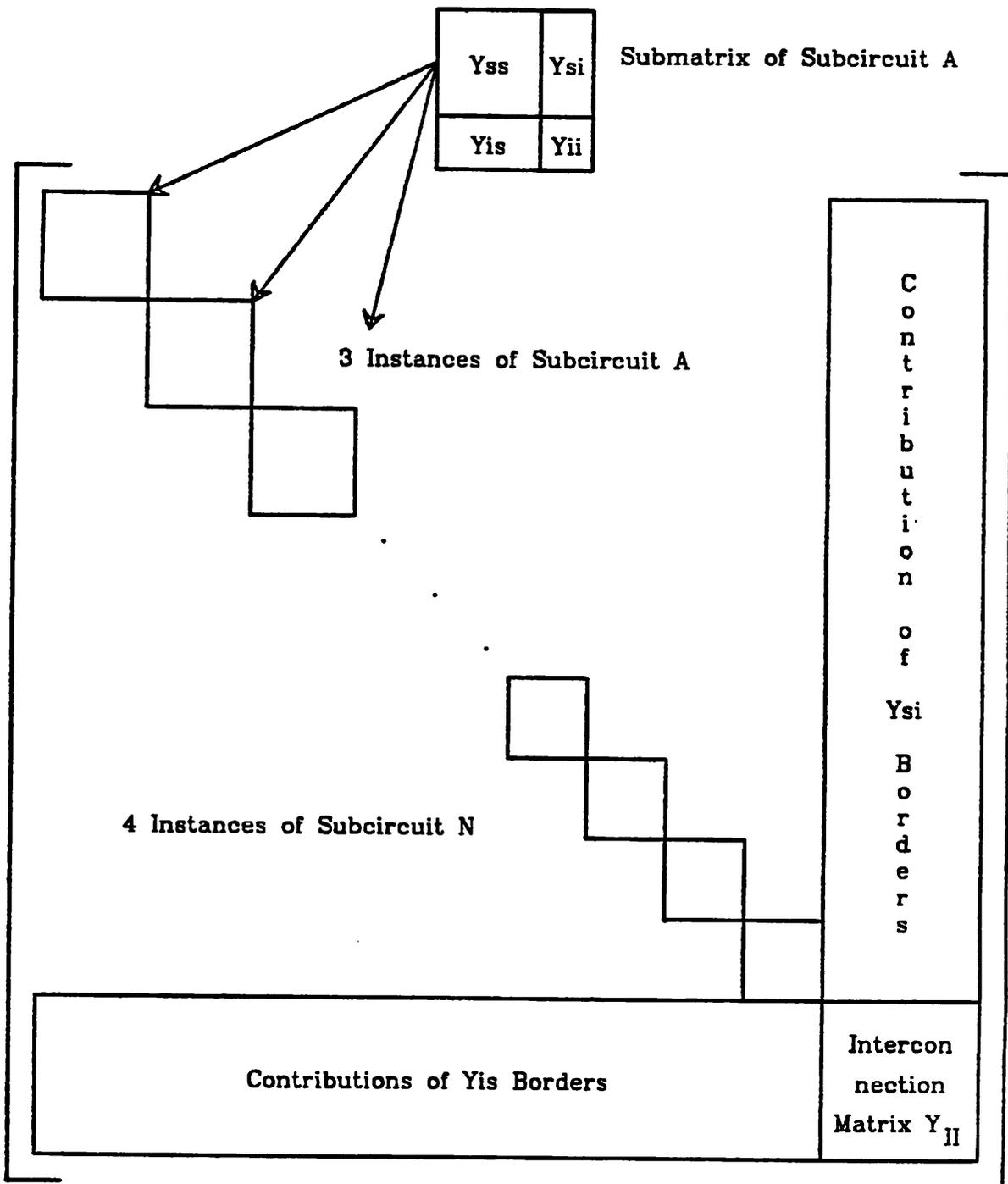


Figure 3.3 Bordered-Block-Diagonal Form of Overall Circuit Matrix

nodal. Then  $Y_{ss}$  represents the nodal equations corresponding to the internal nodes of the subcircuit and  $Y_{ii}$  the equations corresponding to the external nodes. The off-diagonal elements relating internal and external nodes are found in  $Y_{is}$  and  $Y_{si}$ .

The equations corresponding to the internal nodes are unique to each subcircuit instance and can be processed independently of each other. The matrix entries corresponding to the external nodes have to be processed together, generating entries in the interconnection matrix. The overall matrix arrangement as shown in Figure 3.3, can be viewed as a reordering process where all of the internal node parts have been chosen first and the external equation parts have been accumulated with the interconnection circuitry. The border is formed by merging the individual borders of the various subcircuit entries.

The elements  $Y_{ii}^*$  and  $C_i^*$  which are contributed by each subcircuit to the interconnection matrix  $Y_{ii}$  and RHS  $C_i$ , respectively, are found by performing an LU decomposition and forward substitution on the internal part of each subcircuit matrix:

$$Y_{ii}^* = Y_{ii} - Y_{is}(Y_{ss})^{-1}Y_{si} \quad (3.2)$$

$$C_i^* = C_i - Y_{is}(Y_{ss})^{-1}C_s$$

Since

$$Y_{ss} = L_{ss}U_{ss}$$

the nodal equations, Equation 3.1, for the subcircuits can be rewritten as

$$\begin{bmatrix} L_{ss}U_{ss} & L_{ss}^{-1}Y_{si} \\ Y_{is}U_{ss}^{-1} & Y_{ii}^* \end{bmatrix} \times \begin{bmatrix} V_s \\ V_i \end{bmatrix} = \begin{bmatrix} L_{ss}^{-1}C_s \\ C_i^* \end{bmatrix} \quad (3.3)$$

The next step is to solve for the node voltages  $V_i$  of the interconnection matrix:

$$\mathbf{V}_I = \mathbf{U}_I^{-1} \mathbf{L}_I^{-1} \mathbf{C}_I \quad (3.4)$$

where

$$\mathbf{Y}_I = \mathbf{L}_I \mathbf{U}_I$$

Finally the solution for all the internal nodes of the subcircuits are obtained after retrieving the appropriate solution vectors  $\mathbf{V}_I$  from  $\mathbf{V}_I$  and substituting them into Equation 3.3 to obtain

$$\mathbf{V}_s = \mathbf{U}_{ss}^{-1} \left[ \mathbf{L}_{ss}^{-1} (\mathbf{C}_s - \mathbf{Y}_{si} \mathbf{V}_I) \right] \quad (3.5)$$

The terms of the vector  $\mathbf{L}_{ss}^{-1} \mathbf{C}_s$  and matrix  $\mathbf{L}_{ss}^{-1} \mathbf{Y}_{si}$  have already been computed in the subcircuit factorization step, therefore,  $\mathbf{V}_s$  results from a back-substitution process performed independently for all the subcircuit instances.

The linear system of equations is solved by executing generated vector code for the subcircuit matrices and scalar code for the interconnection matrix. This approach is used because otherwise an important part of the total execution time would be spent in equation solution. The data in Section 3.2 also show that generated code has a tremendous impact on the CRAY-1 speedup compared to other computers. One set of vector code is generated symbolically for each type of subcircuit. The same code is then used for the LU factorization and the forward and backward substitution of all subcircuits of the same type. This approach also reduces the severity of the gather/scatter problem due to provisions taken in the design of the data structure, as described below.

### 3.4.2. Semiconductor Device Model Evaluation

The general impression has been that at the level of small to medium circuits most of the CPU time is spent in evaluating the complicated

analytical equations which describe the behaviour of semiconductor devices. It has been equally accepted that for large circuits the linear equation solution time prevails over the device model evaluation. There are several aspects related to model evaluation which require a review of these generally accepted ideas.

The semiconductor device model evaluation in the sense of [Newt77] also includes the loading of all model parameters which is performed for each single device in SPICE2 and the storing of the equivalent conductances (23 for a bipolar junction transistor and 22 for a MOS field-effect transistor). As pointed out in [Cohe81] and [Vlad81a] this memory traffic can account for more than half of the time spent in 'Device Model Evaluation'.

As can be seen in Table 3.3 the percentage time spent in model evaluation vs. equation solution differs among the different processors; the CRAY-1 spends percentage-wise less time in model evaluation due to multiple integer and floating-point functional units and its highly pipelined architecture. With some vectorization of the computation part (SPICEV) the model parameter gather and conductance scatter can take up to 75% of the total model evaluation time [Cala81].

Even for large circuits (over 1000 device-equations) and Fortran solver it is not necessarily true that the equation solution time dominates the model evaluation. This aspect depends on the ratio of semiconductor devices vs. circuit equations as shown in more detail in Chapter 5.

Last but not least, the use of generated machine code for the equation solution can leave the model evaluation the dominant part up to a few thousand devices (actually for all examples analyzed on the CRAY-1). The effectiveness of machine code is a hardware-dependent feature and has

been found to be the highest on the CRAY-1.

After these remarks it is necessary to investigate the possibilities to speedup the model evaluation without any loss of accuracy. The major choices are vectorization, table-look-up models and a different method to include semiconductor device contributions. The first approach is a promising method for a vector computer. It does not exclude the second or third approaches. Table look-up models have been used for some time in timing, [Chaw75], and mixed-mode simulation [Newt78]. Table models have also been implemented in circuit simulators and the maximum reported speedup has been 2. [Tana80], [Shim82]. The larger speedup of 3-4 reported by Newton et al., [Newt77], is due partly to a reduction in the number of equivalent dc conductances which are computed and loaded into the circuit matrix. All the above results with table models in circuit simulators have been obtained on conventional scalar computers.

For the design of CLASSIE, table look-up is not rejected but it is believed that it has a secondary role in relation to vectorization. Table look-up eliminates only the static computation of the current as a function of terminal voltages. As shown in Chapter 5 the evaluation of the Jacobian entries are not a major part of the model linearization. An estimate of the run-time savings which can be obtained using table look-up will be presented based on the simplest available analytical models, e.g., a stripped down Ebers-Moll for BJT's or Shichman-Hodges (LEVEL=1) for MOSFET's [Vlad81b].

A new approach to include contributions from semiconductor devices is possible if the number of equivalent conductances computed and stored for each device is reduced [Newt77]. This includes a redefinition of the linear-

ized device current equation and inclusion of all parasitic effects. An aspect of this is the node suppression technique for parasitic terminal resistances [Lach78]. The trade-off is that more computation is performed in device linearization for keeping the number of equations constant and reducing the storage locations (and time spent in matrix load) by approximately 25% for MOSFET's and 35% for BJT's. Since the equation solution is almost free (see Chapter 5), the node suppression alone is not thought to be too important. The device linearization reformulation has not been undertaken in this project and has been left for a further work.

Model vectorization is analyzed next. Vectorization can be achieved whenever the same set of instructions can be performed on a set, array or vector of data. A vector can be formed by all devices which reference the same model parameters or by all homologous devices in the different instances of the same subcircuit definition. Throughout this text the two ways of grouping devices is referred to as 'across or by models' or 'across or by subcircuits', respectively. A sort has to be added in the preprocessing phase to order the semiconductor devices for vectorization.

The model evaluation has been shown to split into a gather/scatter and a computation phase. If the devices are ordered by subcircuits, both parts can be vectorized. The gather/initialization phase needs data which are independent of each other and describe the different instances of the subcircuit. The conductance scatter stores data in the subcircuit matrices which are unique and separate for each instance as explained in more detail in the following chapter.

When sorted by models solely, the analytical computation part is the only one which can be performed in parallel. This is the only possibility to

take advantage of vectorization of devices occurring at the system (interconnection) level. The explanation lies in the fact that both in the initialization/gather and conductance scatter phases the same set of node voltages or the same (interconnection) matrix are referenced. Thus it is possible that more than one device will try to reference the same memory location at the same time.

The coding style for model evaluation routines is to use different analytical expressions based on the region of operation of the semiconductor device. The equivalent conductance and current sources for a set of diodes are computed by the following Fortran code which uses one or another formulation based on checking the junction voltage.

```

      DO 100 I=1,N
        IF (VD(I).LE.0.0) GO TO 10
          GD(I)=CSAT/VT*EXP(VD(I)/VT)
          CD(I)=GD(I)*VT-CSAT
        GO TO 100
      10      GD(I)=-CSAT/VT
            CD(I)=GD(I)*VD(I)
      100  CONTINUE

```

As already described in Section 2.3.2.3 a DO loop containing conditional statements is vectorized only if all alternatives are evaluated and the result is obtained by a vector merge operation controlled by the logical condition. In the above loop the logical condition is met for negative values of VD(I). At the Fortran level the vector merging is achieved through a call to the function

CVMGx (X1,X2,X3)

where x=T,Z,N,M,P specifies if the variable X3 is checked for TRUE, ZERO,

NONZERO, MINUS, or PLUS. If the condition is met the corresponding element of X1 is stored into the solution vector whereas when the condition is not met the element of X2 is chosen [CRAY80]. The above loop is vectorized by the compiler if coded as follows.

```

      DO 100 I=1,N
          GD(I)=CSAT/VT*CVMGT(EXP(VD(I)/VT), -1.0, VD(I).GT.0.0)
          CD(I)=CVMGT(GD(I)*VT-CSAT, GD(I)*VD(I), VD(I).GT.0.0)
100  CONTINUE

```

There is no clear win with vectorization. On one hand there is a clear speedup in a vectorized computation due to the reasoning given above but on the other hand the computer has to perform the double or threefold work to evaluate all the alternatives. If both the static characteristic and the charge storage elements could be obtained from tables there would be the advantage of not doing unnecessary work as in the analytical case.

### 3.4.3. Memory Access Considerations

As mentioned earlier the high speed of floating-point computation on the CRAY-1 makes memory access an important issue. A data structure which neglects this aspect can ruin the performance and make the program spend most of its time in data transfers to and from memory.

Several peculiarities about the CRAY-1 computer which must be considered are outlined next. The load or store of a floating-point number to memory requires four clock cycles. This number can be in average slightly higher in a program because the instruction requires an address register to be set up as well for the memory transfer to take place. It can be assumed that five clock cycles is a typical number. If data are transferred as a vector load or store operation, it takes only one clock cycle for each data,

provided that the same memory bank is not accessed (the CRAY-1 memory is organized as 8 or 16 memory banks). The vector length must be at least two to offset the start-up time of the instruction.

The problem in performing vector load and store is that the hardware instruction can operate on data which are stored sequentially in memory or are a constant stride apart. The sparse-matrix load operation (or conductance scatter) needs the capability of indirectly defining vectors or equivalently to gather or scatter data from or to random memory locations to or from a vector register. The characteristic sparse-matrix load loop is

```

      DO 10 I=1,N
          MATRIX (INDEX(I)) = MATRIX (INDEX(I)) + TERMS(I)
10     CONTINUE

```

There are two ways of reducing the time in memory transfers for this operation. First, at the interconnection matrix level one can either use the gather/scatter routines of the CRAY-1 library (20 clock cycles per element) or generate machine code (12 clock cycles per element). Second, for the hierarchical analysis of CLASSIE it becomes a matter of data structure definition to make vector gather and scatter possible at the subcircuit level where each instance of a cell is represented by its own MNA matrix.

The dynamic memory management scheme of SPICE2, [Coh76], is preserved in CLASSIE for handling the circuit data. However, the data structure which is predominantly of the linked-list type in SPICE2 is changed. The data structure and the main analysis loop are designed to accommodate the new algorithms and to make the vectorization possible. Thus, each element type has its own parameter table and data may be aligned in such a way that vector mode operations can be programmed most effectively. The information on the circuit sparse matrix is organized as one set of sparse submatrix

pointers for each type of subcircuit and another set of pointers for the interconnection matrix. The nonzero subcircuit matrix entries and RHS for each cell of the same type are aligned and appear as one data set.

## CHAPTER 4

### Program Description

#### 4.1. Introduction

The algorithms and techniques used in CLASSIE for the efficient simulation of LSI circuits are described in this chapter as part of a detailed program presentation.

A most important component of a large program is the data structure used to encode the description of the specific problem to be solved. The data format should be well suited for both the algorithms and the computer specifics. The detailed description of the data structure used in CLASSIE which suits both the hierarchical aspect of the LSI circuit problem and is efficient from the computer architecture perspective is given in the first section. The memory management package which supports the selected data structure is also presented.

The Newton-Raphson iterative scheme has been used successfully in many simulators. The details of its implementation in the context of a two-level analysis are described in a separate section. A further section describes the vectorized model evaluation for semiconductor devices at the subcircuit and system level.

The representation of an LSI circuit consists of a large set of equations. In the case of a two-level analysis the equations are scattered over many small matrices. The small matrices represent the cells of the LSI. One

matrix gathers all the information about the system behaviour and each cell (subcircuit) matrix adds its contribution. The management of these matrices and solution of the corresponding equations is the subject of a major section.

## 4.2. Data Structures

### 4.2.1. Description of the Data Structures

The linked lists of integers and reals used in SPICE2 to store the circuit element information is quite inadequate for large circuits. The first reason has already been mentioned in the previous chapter and pertains to vectorization features on the CRAY-1. Another argument is that on a conventional medium-size computer with virtual memory, the number of *page faults* can increase drastically for a circuit with a few hundred elements because of the random locations of the elements linked to one another. In a virtual memory computer there is a physical limitation on the main, fast access memory. Certain parts of the program and data address space needed for the current execution are kept in the main memory as sets of pages which contain a predefined number of addresses. Each time a virtual address is needed which can not be found in the pages currently in main memory a page fault occurs and the computer has to copy the program-requested address space from a mass storage device.

In CLASSIE an integer and a real table is included for each element type. At the present time there are 36 'elements' which can be categorized as:

i) circuit elements, resistors, capacitors, inductors, mutual inductors, linear current and voltage controlled current and voltage sources.

independent voltage and current sources, diodes, bipolar junction transistors, junction field-effect transistors and MOSFETs.

ii) model elements which contain the sets of parameters characterizing the analytical semiconductor-device models implemented in the program.

iii) subcircuit definitions and subcircuit calls which provide new attributes compared to SPICE2 due to the hierarchical, subcircuit-oriented analysis.

iv) output variables which store the information on voltage and current variables and the mode of analysis to be saved.

v) print/plot elements (information).

The resistor integer and real table structure are shown in Figure 4.1. It can be seen that the same information as in SPICE2 is stored in this table and the description contained in the SPICE2 Reference Manual [Coh76] is generally valid. The main differences are that the pointers stored, e.g., to the location of the corresponding real data (LOC+1), are offsets rather than absolute addresses in the respective table (IRVAL).

The two-level analysis requires a provision in the data structure for a two-way link between subcircuit definition and subcircuit instantiation. It is also necessary to have pointers to the different elements which are part of the definition.

As shown in Figure 4.1 the (LOC-1) position of an element integer table contains the offset in the subcircuit definition table (ISSNOD) of the definition which it is part of. If the element has been introduced at the system level then this location is zero. Each cell instance is described by its own MNA matrix which has the same sparsity pattern for all instances of the same definition. Thus all the information contained in the integer table is

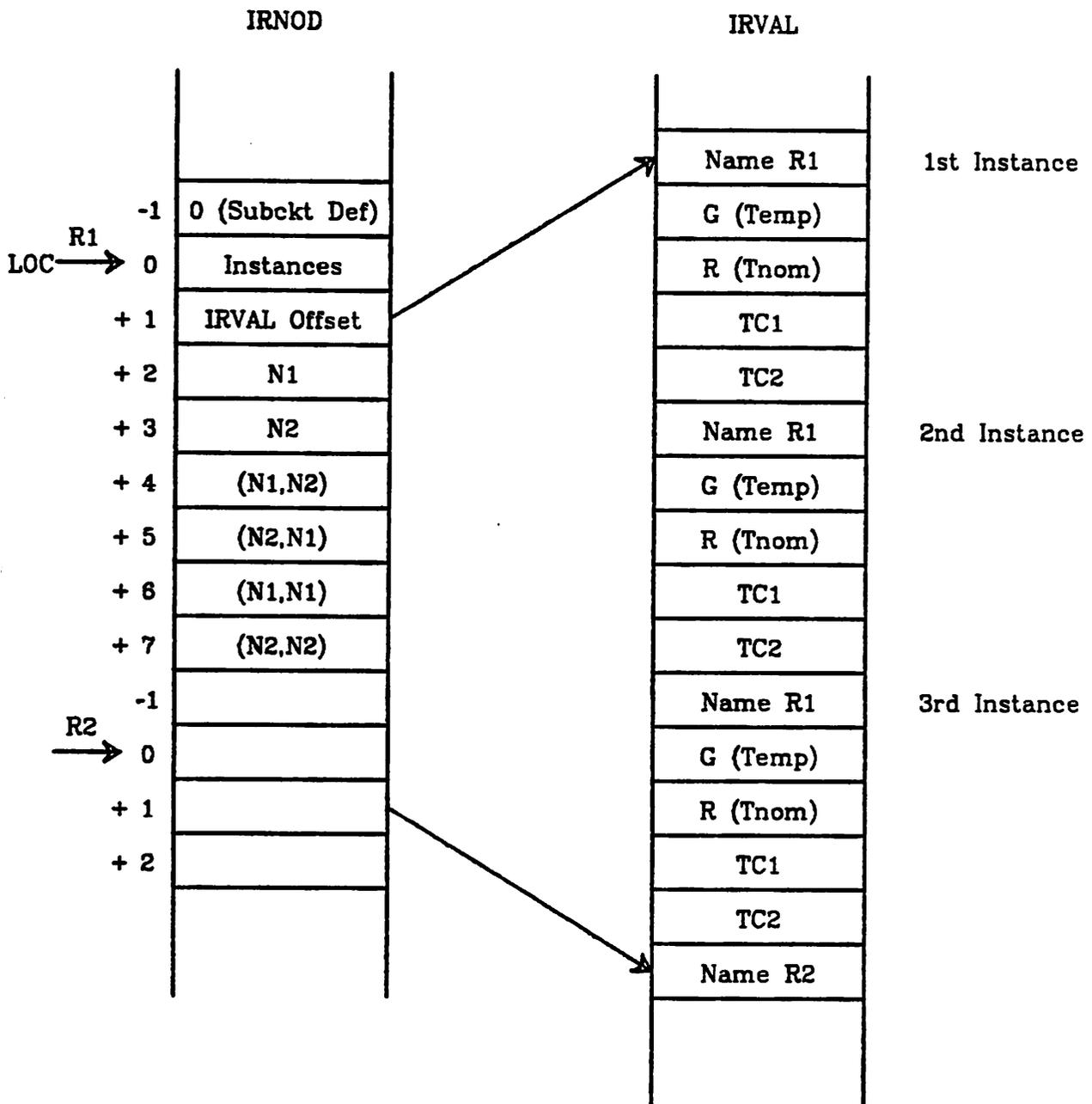


Figure 4.1. Resistor Integer and Real Table Structure

valid for all the instances of a particular cell definition. Multiple copies of the real tables are necessary because a certain element can assume a different value in a different instance. Since there is a pointer to the location of the reals for the first instance it is necessary to store the number of instances of the cell. This number is kept in location (LOC) and is equal to 1 if the element is described at the interconnection level. Other element types, e.g., capacitors or semiconductor devices which have charge storage associated, have a pointer to the corresponding charges in the LXi tables [Cohe76]. In the case of subcircuit elements the pointer to the LXi table is similarly to the one pointing to the value table, a pointer to the first instance.

The integer tables describing circuit elements in SPICE2 contain pointers to tables which store polynomial coefficients, as is the case with nonlinear controlled sources, or waveform characteristics, e.g., the transient description of independent sources. The number of these 'secondary' tables can grow considerably and become a memory management problem if the number of elements which generate them gets large. Therefore CLASSIE recognizes only linear controlled sources. For the independent sources five arrays have been defined, ISRPUL, ISRSIN, ISREXP, ISRPWL and ISRSFF, which store the waveform characteristics for the five types: pulse, sinusoidal, exponential, piecewise linear and single-frequency. What formerly was yet another table pointer in SPICE2 in the independent source table is replaced by an offset in the appropriate waveform table.

All information relative to the cells is kept in the subcircuit definition table which acquires a special importance. Figure 4.2 gives a graphic description of the information available in this table. There are offsets to

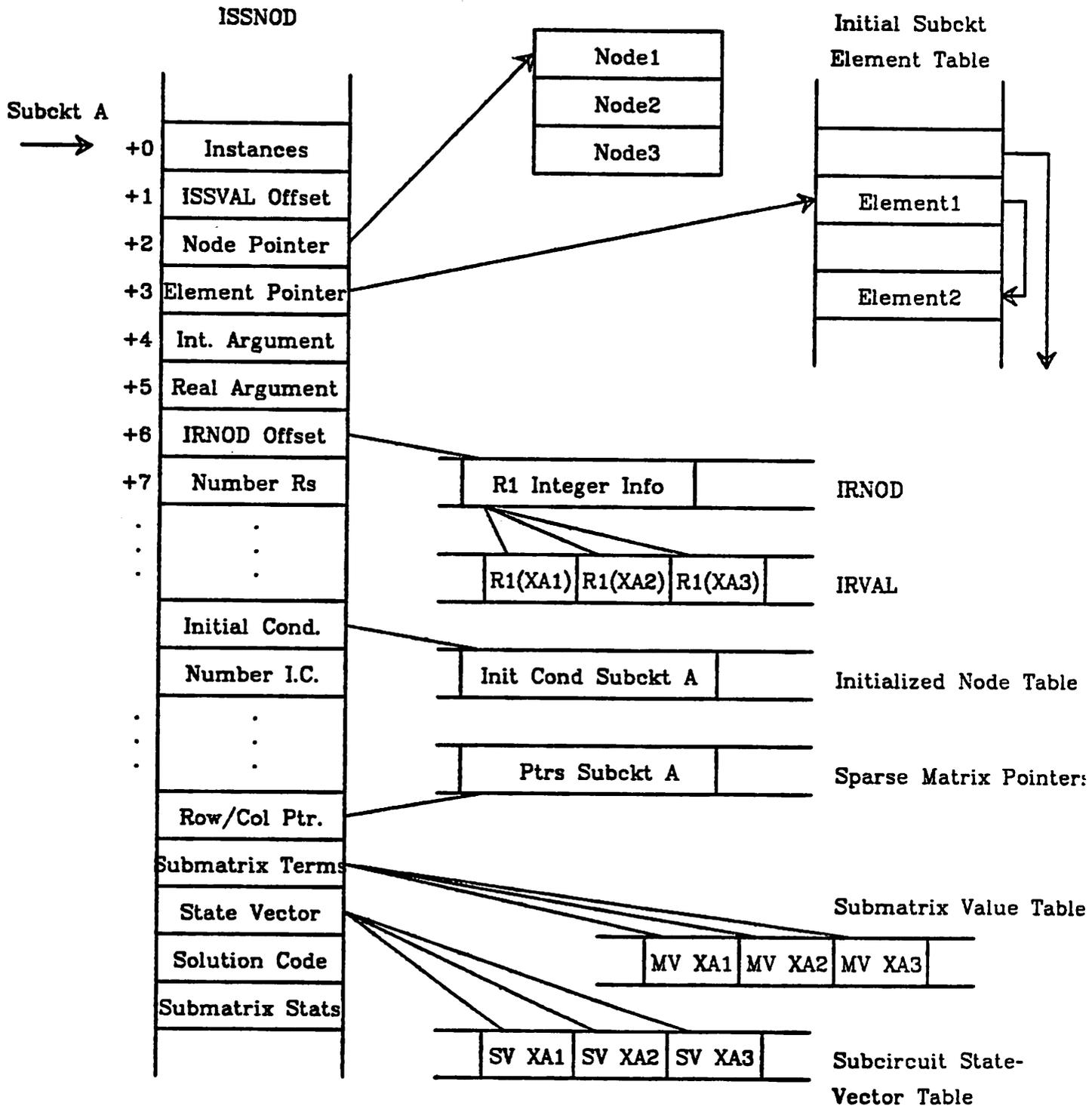


Figure 4.2. Subcircuit Definition Table Structure

the first of the elements of each type which are part of this definition and the count. Other offsets refer to all tables where integers (sparse matrix pointers) and reals (charges, node voltages or matrix entries) for the cells of this definition are stored. The multiple copies stored in the tables of reals are aligned at constant strides which are also saved in the subcircuit definition table. Other constants and offsets represent information such as the number of instances, the number of internal and external nodes, and the address of node-list table or initial condition offset. A complete description of the subcircuit definition and subcircuit call table are given in Appendix 3.

A very important issue with the new data structure is the special care which must be exercised with tables which contain pointers (absolute addresses) to other tables. Such tables pointed to from other integer tables are the node table in subcircuit definition and subcircuit call, or the matrix location table in the subcircuit call table (see also Appendix 3). Several modifications have been made in the memory management program and two new routines have been added to accommodate these features, as is described in the following section.

In the analysis part of the program several integer and real arrays are used to maintain the information about the sparse-matrix structure and the matrix entries. In CLASSIE there is a duplicate set of all these tables: one storing the data for the subcircuits and the second for the interconnection circuitry. Thus, IRSWPF, IRSWPR, ICSWPF and ICSWPR contain the forward and reverse mapping of internal node numbers into equation numbers for rows and columns, respectively; ISRSWF, ISRSWR, ISCSWF and ISCSWR have the same data corresponding to the subcircuit nodes and equations. The

same approach is used for the matrix entry tables LVN and LSVN, and the charge-storage tables LXi and LSXi, etc. In the subcircuit description tables enumerated above the entries for different subcircuit definitions are stacked onto one another and the pertinent offset for the first instance (for reals) is stored in the subcircuit definition table.

#### 4.2.2. Memory Manager Functions

The SPICE2 program uses a set of routines which constitute the 'memory manager' to implement and maintain its data structure within a block of memory which can be varied dynamically during program execution or is fixed depending on the host computer. The same memory manager routines are used in CLASSIE and the storage area is increased or shrunk dynamically during execution.

Several modifications have been found necessary in the SPICE2 memory manager in order to meet the needs of CLASSIE and large circuits. The SPICE2 memory manager [Cohe76], [Dowe80], does not provide any capability for tables which contain table pointers. In SPICE2 this feature is avoided by the fact that all the element data (which can contain pointers) are part of a single table (IELMNT) which is allocated first and can not be moved. The creation of individual tables for each element type requires the capability of checking for table pointers within tables and of updating their information when blocks are moved around. The modifications brought to the memory manager of SPICE2 are described in the following paragraphs. The basic functions and subroutine names are preserved as described by Cohen [Cohe76].

The table-entry table which contains in SPICE2 five entries, offset within data memory area, block size, actual number of words used, address of table pointer and number of words per entry, has been augmented by two more. The first gives the number of words allotted to a table which are not used presently but is expected to be needed later [Cohen81]. This is called slop memory which prevents excessive moving of memory when more than one table is grown at the same time. This feature, which has been added during the development of the SPUDS program, is not as necessary in CLASSIE because the table size is grown in exactly precomputed amounts rather than by one element at a time. The second new entry is the number of table pointers within a table.

Every time a new block is allocated or extended a compression of the data memory is performed by subroutine COMPRS. This routine has been modified to check each block that it moves for internal table pointers. If such pointers are found their address is updated in the table-entry table of the blocks they point to. Checks for internal pointers have been added also in RELMEM, which releases part of a block to the memory manager, and in CLRMEM, which releases a table. An error condition is flagged if the released area contains any pointers.

In certain situations it is necessary to release a certain area of a table which contains pointers or copy a certain part with pointers to a different table. The new routines which perform this task are called DELETB and COPYTB. These operations are necessary when certain subcircuit definitions or subcircuit calls have been expanded and must be deleted or others must be copied to a new table.

Another characteristic for large circuits is the number of tables to be managed which is in the range of a few hundred to a few thousand. Function MEMPTR is called by all memory-manager routines to check if the table under consideration exists and to find the location of its table entry table. In the initial stages it has been observed on SPICEV that MEMPTR claimed the largest part of the 8-14% of the total analysis time spent by memory manager operations. The contribution of MEMPTR has been reduced significantly after changing the linear search algorithm used before by a binary search.

#### **4.3. Newton-Raphson Iterative Scheme Implementation**

The Modified Newton-Raphson scheme has proven to be the most reliable solution method for a set of nonlinear algebraic equations which are the mathematical representation of an electric circuit after time discretization. In the case of the hierarchical simulation implemented through tearing decomposition in CLASSIE the Newton-Raphson iterations are performed at two levels: at the cell (subcircuit) level and the interconnection circuit level. These two processes are independent of each other as is the processing of the different subcircuit definitions from one another. The two levels of the analysis communicate only for loading the equivalent conductances of the subcircuits into the interconnection matrix and in the solution phase for retrieving the solution at the external nodes to solve all the other nodes of the subcircuits. The general iterative scheme for the two-level analysis of CLASSIE is shown in Figure 4.3.

A blown-up view of the linearization and matrix (subcircuit and interconnection) load algorithm is shown in Figure 4.4 in a high-level language-

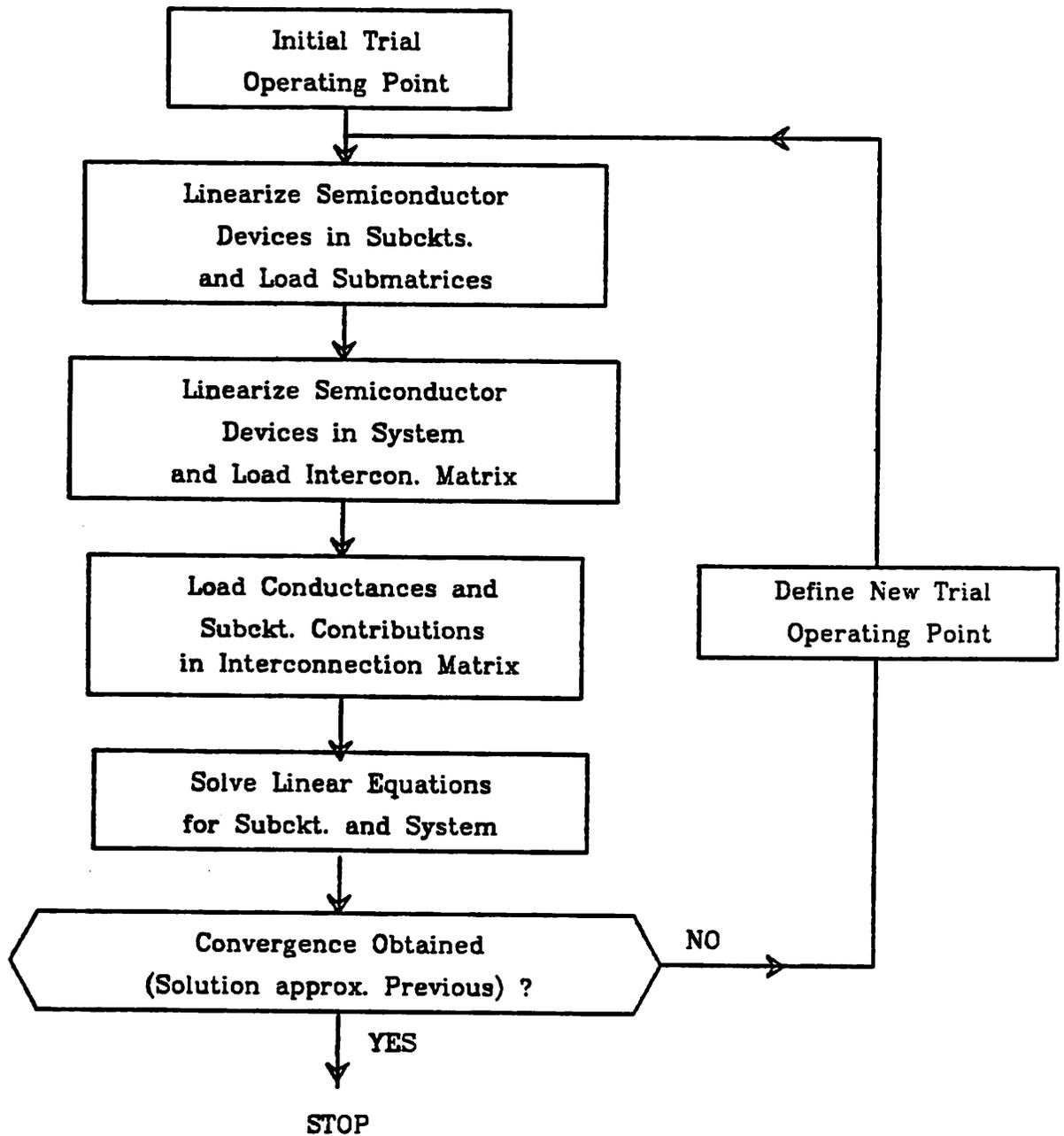


Figure 4.3 Two-Level Iterative Newton-Raphson Solution Algorithm in CLASSIE

like description. The linearization of semiconductor devices is first performed for all subcircuit instances in the system and all the conductances are loaded in the subcircuit matrices. All the computation and gather-scatter processes can be vectorized across subcircuit instances. For an important speed improvement the more instances the better; this has been given as one of the criteria in the selection of the subcircuit size in Section 3.3 of the last chapter. The descriptor 'Vfor' has been introduced to represent a vectorized loop. This process is repeated for all the different subcircuit definitions.

Both linear and nonlinear elements occur not only at the cell level but also at the system level. All these elements are loaded next. The gather of data from memory and the scatter of the equivalent conductances back to memory is a purely sequential process which can not be vectorized because of the possibility of two elements generating entries in the same location. The computational part of the semiconductor device linearization is vectorized across models.

The interaction between the cells and the system takes place through the conductances at the external nodes. These can be actual conductances or fill-ins generated after the setup of the sparse matrix structure. These contributions are loaded in a purely sequential manner in the interconnection matrix. As can be seen in the data structure description of Appendix 3 there is a table of 'matrix locations' pointed to by both the subcircuit definition and the subcircuit call. The first contains the locations in the subcircuit matrix while the second the locations in the interconnection matrix where the data from the former matrix are stored for each subcircuit call. This process is time consuming and accounts from 5-10% of the

total analysis time as explained in more details in the following chapter.

The linear equation solution takes also place at two levels. Conceptually the algorithm outlined in Section 3.4.1 can be translated in the program description shown in Figure 4.5. This description can be followed on the BBDF matrix shown in Figure 3.3. A vectorized LU (VLU) and forward substitution (VFS) is performed on all instances of a subcircuit definition. The vectors are defined across the instances of the presently processed subcircuit definition, i.e., when element  $L_{jk}$  is evaluated it is done so for all submatrices having the same topology. In each submatrix this process is performed for the first  $N_{int}$  equations, where  $N_{int}$  is the number of internal nodes. The interconnection matrix  $Y_{ij}$  is also updated, if there are elements  $Y_{ij}^*$  being modified as a result of operations between terms of the border, according to Equation 3.2. The right-hand side terms which correspond to the external nodes in the subcircuit and are denoted by  $C_j$  in the previous chapter, Equation 3.1 and Equation 3.2, are accumulated in the corresponding RHS terms  $C_j$  in the system matrix.

A full solution is then performed on the system equations, and the part of the solution vector is scattered to the solution vectors of the subcircuits. The last operation is the vectorized back substitution in the subcircuit equations which results in all the internal node voltages of the cells.

A last major step in the Newton-Raphson analysis is to check for convergence of the solution. The convergence test consists of two checks: one is done on the terminal voltages and currents of the nonlinear devices and the second on all variables of the solution vector (node voltages and currents) of the circuit. The test checks if the variables at the last iteration are within a desired error from their value at the previous iteration. If the

```
for all SUBCKT DEFS {  
  Vfor all instances {  
    load linear elements  
    linearize and load semiconductor devices  
  }  
}  
  
load interconnection linear elements  
linearize and load semiconductor devices  
  
for all SUBCKT DEFS {  
  for all instances {  
    scatter border (external nodes)  
    into interconnection matrix  
  }  
}
```

Figure 4.4. Subcircuit and Interconnection Matrix Load

```
for all SUBCIRCUIT DEFINITIONS {
  Vfor all instances {
    LU decomposition
    if both L, U in border {
      for all instances {
        update corresp. interconnection term
      }
    }
  }
  Vfor all instances {
    FS forward substitution
  }

  for all instances {
    update RHS of interconnection
  }
}

# Solve interconnection matrix
LU decomposition
FS forward substitution
BS back substitution

Vfor all instances {
  BS back substitution
}
}
```

Figure 4.5. Linear Equation Solution

nonlinear devices fail then the node voltages are not further checked. Although this seems to be a simple operation it can become an important part of the total analysis time for a large circuit where the number of the variables which are checked is in the few thousand range.

In the solution vector test an additional mapping operation must be performed between the values of the variables at the current iteration which are in the matrix column order (different from the row order due to the nonsymmetry of the MNA) and the previous values which are stored according to the program internal node numbering. The two mapping operations are reduced to one by storing an additional indexing array INIRHS for the interconnection equations and INXRHS for the subcircuits.

The interconnection solution is always checked first. Only if it has converged, then, the internal variables of the subcircuits are tested. This check is done in vector mode at a rate of one operand access per clock cycle (12.5 ns). The results of the convergence test are stored in a vector RNONCN as ones in those elements which correspond to subcircuits for which the solution at the node currently checked has diverged and zeroes in the rest of the elements. A fast CRAY library routine,

`SSUM (N, ARRAY(1), ISTRIDE)`

which sums the first N elements, ISTRIDE locations apart, of an array ARRAY, is used for counting the subcircuits which failed the convergence test. All the above provisions keep the time spent in convergence checking below 10%.

#### 4.4. Vectorized Semiconductor Device Model Evaluation

This section presents the approaches used in CLASSIE to get the maximum speedup in the semiconductor device linearization on a vector computer. This part of the simulator has been identified in the previous chapter as using more than half of the analysis time of a medium circuit. The importance of the semiconductor model evaluation increases as the percentage time spent in the linear equation solution is diminished using machine instructions. There are a number of tasks performed as part of what is generically known as model evaluation. The different tasks can be visualized in Figure 4.6 which shows a description of a model evaluation routine(s) as implemented in CLASSIE.

In CLASSIE as in SPICE2, semiconductor devices are described by geometrical features which are individual for each device and general parameters, e.g., saturation current,  $I_s$ , gain factors,  $\beta_F$  and  $\beta_R$ , for a bipolar transistor, and threshold voltage  $V_T$ , thin oxide thickness,  $t_{ox}$ , for a MOSFET. The general parameters are the 'model' parameters and are stored in model tables. There is a 'model element' type for each device type, i.e., diode, BJT, JFET, and MOSFET. The first thing which the model routine has to do is to obtain the model parameters from memory. BJT's are described by 55 and MOSFET's by 46 parameters, to mention only the two most often referenced device types. This step is called model-parameter gather. After each device is linearized a scatter operation takes place, which stores the device indefinite admittance matrix into the circuit matrix. As mentioned earlier, these two operations account for more than half of the time spent in the model routine.

The importance of the model-parameter gather is made negligible in CLASSIE based on the fact that more than one device uses the same model parameters. In the setup phase a device reordering takes place.

1. All semiconductor devices which are introduced at the system level are stored first in the device table; the devices which reference the same model are grouped together. A pointer to the first element of a group and their number are stored in the model table.
2. The devices introduced as part of cells (subcircuits) are stored next, ordered by cell definition. Inside this group they are ordered also by models. The number of devices of the same model is stored this time in the table of the first device of each group.

This reordering operation takes place only once for the entire simulation. The model parameters need thus be gathered only as many times as there are model definitions.

The actual computation of the equivalent conductances of the linearized model goes through several stages as can be seen in Figure 4.6. All devices referencing the same model are stored contiguously and can be linearized in only one pass through the code using vector computation. For this purpose a buffer table is created which stores all equivalent conductances, currents and other intermediate results for the matrix scatter. In order to keep this storage space to a minimum a vector length of 64 is provided for each variable. This matches the maximum vector length of the CRAY-1. If there are more devices than 64 the whole loop is repeated under program control, see Figure 4.6, using the same buffer space.

The routine as depicted in Figure 4.6 is called at least twice at each iteration, once for loading the transistors of the interconnection circuit and

```
for all device models
  gather model parameters
    for every 64 devices {
      initialize state-vector (XINIT)
      limit new terminal voltages (XLIMIT)
      Vfor all devices {
        compute new conductances
        if transient analysis
          compute charges and conductances
        check convergence on state vector
      }
      scatter indefinite admittance matrices (XMATLD)
    }
  }
```

Figure 4.6. Semiconductor Device Model Evaluation Routine

once to perform the same task for each subcircuit definition. An argument of the routine is used for passing the information of the hierarchy level at which model linearization is performed. Different routines are called for initialization and matrix load depending on the value of this argument. In Figure 4.6 the names written in parentheses next to each task denote generic names of routines.

The specifics of XINIT, XLIMIT and XMATLD are described next. XINIT initializes the terminal voltages of each device based on analysis status and device parameters. Most often the terminal voltages are computed from the node voltages at the last iteration. The resulting terminal voltages are stored in the buffer table at a constant stride apart so they can be used in vector operations. The operations of XINIT can be vectorized only in the case when the node voltages are stored correspondingly. This is true for subcircuits where the individual copies of node voltages are contiguous. There are two versions of XINIT, a scalar and a vector version, called alternately depending upon the hierarchy level to which the transistors belong.

XLIMIT assumes different names, e.g., PNJLIM, FETLIM, but performs the same function, that of limiting the change in device terminal voltages between iterations. All data involved in this operation are stored in the vector buffer so the routines are fully vectorized. For the best time performance the routines XLIMIT check first if all terminal voltage variations are bound by a certain error and quit if this assumption is true. This test saves a large amount of computation involving logarithmic and exponential functions.

XMATLD stores the linearized conductances and currents in the matrix and RHS. The locations (23 for BJT's and 22 for MOSFET's) where the device

indefinite matrix terms are stored, are obtained from the integer device tables. This function has also a scalar and vector version which are used for the interconnection or subcircuit matrices, respectively. All transistors which have been linearized in vector mode generate random entries in the interconnection matrix, sometimes different conductances of different transistors or even of the same transistor are accumulated in the same location. Although the matrix load operation is referred to as a scatter operation it needs an intermediate vector which in turn can be scattered to memory. This intermediate vector accumulates the elements of the indefinite matrix of each device such that no conflict occurs.

An important issue is the definition of vectors throughout the model evaluation. For the transistors at the system level it is quite straightforward to define a vector across all devices which reference the same model. As has already been mentioned only the computation part is vectorized for these elements. The main tables where vectors are defined are the buffer and the LXi tables which store the state vector.

The different possibilities to define vectors can be presented best by the following example. Assume that a circuit contains 12 instances of a subcircuit OPAMP which in turn has 20 MOS transistors. The semiconductor devices at the subcircuit level can define a vector across all instances of that cell. Thus, the transistors named M09 in all 12 instances of the subcircuit OPAMP are linearized in one pass through the code. This results in 20 passes through the model evaluation code with a vector length of 12 each time. Although all tasks can be vectorized in this approach a longer vector can be used in the computation phase where all transistors of the same model and for all instances of the subcircuit can be grouped together. The

operation flow remains the same as shown in Figure 4.6. In the initialization and scatter tasks the longer vector used in computation is divided into a number of short vectors which contain as many elements as cell instances. The gain in this approach comes from a reduction in start-up times due to longer vector operations. In the above example assume that 5 of the 20 transistors are depletion loads and are characterized by the same model parameters and that the remaining 15 are enhancement devices and are also described by a unique model. For the 12 instances the execution of the computation loop is reduced from 20 times to 4 times (once for the depletion devices and three times for the 180 enhancement devices) for a maximum vector length of 64.

Another trade-off in the design can be between a longer vector loop which performs also more computation than necessary or a number of shorter loops to which the execution is directed depending on analysis status flags. Both approaches lead to almost similar speeds.

As a final comment, the convergence check of the semiconductor devices is performed in the same manner as for the node voltages. The terminal voltages and device currents are compared in a vector loop and a vector with ones for the diverging elements and zeroes for the converging ones is set up. The fast vector accumulation library routine, SSUM, is then used for a fast result.

#### **4.5. Linear Equation Solution**

This section describes in more detail the two-level circuit matrix solution process from equation setup to the generation and execution of vector and scalar machine code. The machine code has proven the most efficient

way of taking advantage of the CRAY speed and architecture.

#### 4.5.1. Processing in the Setup Phase

The two-level analysis uses two sets of sparse-matrix pointers and arrays, one for the subcircuits and one for the interconnection as already mentioned in Section 4.2.1. The linked list description of the matrix has four tables, as described in [McCa76]. There are four more mapping tables used to record the correspondence between independently swapped rows and columns and the internally renumbered nodes. All these pointers are initialized in the setup phase and then defined for the rest of the analysis during the first pass through LU decomposition.

A number of routines, MATPTR (MATPSS), RESERV (RESRSS), trace the lists of the elements at the system or cell level, respectively, and reserve matrix locations, store these locations in the element integer description tables and set up the sparse matrix pointers. Memory management problems have been noticed in SPUDS, [Cohe81], in this part of the program since all four sparse-matrix pointers grow by one element every time. In SPUDS this has been fixed introducing the slop memory with each block. In CLASSIE the four blocks mentioned above are extended when necessary by amounts equal to the number of equations. This approach is the most efficient since it avoids the call of several memory manager functions tens of thousand of times in the case of large circuits.

Two aspects of reordering are performed at this time. Both are topological in nature. First, the current equations are swapped with the corresponding voltage equations since it is known that the former introduce zeroes on the diagonal. Second, the equations corresponding to external

nodes are reordered in the subcircuit matrix to be last. This reordering is necessary for the partitioning of the processing which takes place at the cell level from that at the system level. Thus, LU decomposition can be completed for the subcircuit matrix (internal nodes) first. Then the terms generated in the lower right corner (external nodes) can be scattered into the interconnection matrix. The entries which link the cells to the system are also reserved at this time.

Storage space in the LXi tables for charges associated with each element is reserved at this point as well. For the subcircuits two alternatives have been implemented so far. The first reserves space for each subcircuit instance and aligns data for different instances back-to-back. This approach did not prove satisfactory however if long vectors are defined as described above. For this latter case the locations for charge components in the LXi tables must be allocated consecutively for all instances of each semiconductor device. This arrangement is dictated by the vector memory access (gather/scatter of charge values during the iterative analysis).

After all matrix entries have been reserved and the topological reordering performed a set of four routines, MATLOC (MATLSS), INDXX (INDSS), establish the location in the linearly stored sparse matrix array for each contribution of each element. These data are stored in the element integer tables for use in the matrix load which takes place at each iteration. The locations of matrix entries which link the two levels of analysis together are also established. Their number is a variable and thus are stored in a separate table which is pointed to by either the subcircuit definition table for the cell matrix locations or the subcircuit call table for the interconnection matrix locations. These tables grow after numerical pivoting because of

fill-ins.

#### 4.5.2. Reordering

The equation reordering for a well-conditioned matrix during the solution is performed in two steps. The topological reordering is done in the setup stage; a numerical reordering is performed using partial pivoting after the first load of the matrix. At this step is also found the fill-in structure for each matrix. A more detailed analysis of the reordering process and matrix statistics for the system and cell equations is presented in the following two sub-sections.

##### 4.5.2.1. Interconnection Matrix

The reordering for the interconnection matrix is similar to the one performed on the entire matrix in SPICE2 since pivoting has been introduced [Vlad78]. In the partial pivoting strategy the potential pivot is compared to all the non-zero entries in that column. An entry is accepted as a pivot if

$$|\text{pivot}| \geq \text{PIVREL} \times \text{colmax} \quad (4.1)$$

where,

$$\text{colmax} = |\text{maximum column entry}|$$

$$|\text{colmax}| \geq \text{PIVTOL}$$

It is checked first that the maximum entry in that column 'colmax' is larger than an absolute minimum, PIVTOL. Candidates for pivots are first looked for only on the diagonal. If none can be found, then the whole matrix is searched. If at any point there is no element larger than PIVTOL the matrix

is declared singular. Both PIVTOL and PIVREL are OPTION input parameters for SPICE2 and CLASSIE. This reordering is performed only once during a circuit analysis unless an ill-conditioned pivot is discovered at a later iteration during the simulation. In the remainder of the analysis the pivot is checked only against PIVTOL. If the Fortran solver is used, an 'on the fly' pivot change can be performed and the solution continued from that point on. The approach is slightly different if machine code is used as outlined in a later section of this chapter.

Several interesting comments can be made on the interconnection matrix statistics. It has been anticipated that this matrix is mostly full [Cala80] and a vectorized solver can be used. Table 4.1 presents the matrix statistics of a few medium and large circuits which have been used as benchmarks for CLASSIE. The number of equations, number of non-zeroes, number of operations and percent sparsity are presented for SPICE2 and CLASSIE. There are no SPICE2 entries for the subcircuits NAND and OPAMP because of a lack of meaning; subcircuits are expanded in SPICE2 and cannot be singled out from the individual elements of the overall uniform circuit. It is noteworthy that the large matrix in SPICE2 is replaced by a smaller interconnection matrix and a small subcircuit matrix occurring as many times as there are instances. Even if the interconnection matrix is not as sparse as the overall circuit matrix in SPICE2 it is in all cases more than 85% sparse. This means that the fastest way of solving this matrix is still by scalar machine code rather than by a vector block solver [Cala80]. Such a block solver can be used for matrices with high local densities which is not the case in the circuit area, not even when a two-level analysis is performed. For the adder circuits which are bipolar and built only of NAND gates most terms in the interconnection matrix originate from fill-ins. In

SPICE2					
Circuit	Eqn.	Terms	Fills	Ops.	%Sparse
Adder4	451	2663	628	6062	98.69
Adder16	1747	10677	2594	24560	99.65
Filter	410	4069	710	8290	97.58
CLASSIE					
Adder4	55	389	328	1579	87.14
Adder16	163	1419	1248	5907	94.66
Filter	157	1652	260	5298	93.3
NAND	16	95	36	345	62
OPAMP	16	140	34	708	45

Table 4.1. Interconnection and Subcircuit Matrix Statistics

the case of the MOS switched capacitor filter (Filter) which is built from opamps at the subcircuit level and MOS switches at the system level the percentage of fills is the typical one.

#### 4.5.2.2. Subcircuit Matrix

The reordering issue is more delicate in this case as compared to the interconnection matrix. All instances have to preserve the same topology so an optimal order must be found. In the case where the program default initial conditions are used for all devices, the entries in all instances are the same and the order which fits one is good for all. However, if user-specified initial conditions are used, the matrix entries vary widely among the different instances. The pivoting algorithm described by Equation 4.1 is still valid with the difference that 'colmax' is taken also across instances. The instance which has been initialized might have some transistors turned on while the instances which use the default initialization have them all off in the case of MOSFET's. The relative tolerance test described in Equation 4.1 might stop the analysis because the matrix entries for some instances are much smaller than the same entries for user-initialized instances.

The Filter benchmark circuit is such an example. For convergence purposes it needs certain opamps to be initialized. The parameter passing option is used to initialize certain internal nodes without increasing the number of subcircuit external nodes as in SPICE2. The relative magnitude test must be turned off to keep the analysis going. This is easily done by specifying  $\text{PIVREL}=10^{-12}$  on a .OPTIONS statement.

### 4.5.3. Machine Code Solution

The solution of a set of sparse linear simultaneous equations requires a search of the sparse-matrix pointers for the necessary operands in addition to the arithmetic operations. As shown in Chapter 3 this search can account for a larger or smaller part of the linear equation solution depending on computer architecture and circuit complexity. The original idea of a search-free code goes back to Gustavson [Gust67] who proposed a two-step process consisting of a symbolic solution generating a Fortran sequence which needed recompiling and could then be used for solving the given system of equations without any search. The most useful approach to circuit simulation has been introduced by Cohen in SPICE2 [Coh76]. It consists in a symbolic phase which generates machine instructions during the first solution; the machine code can be executed directly at subsequent iterations.

The two-level analysis of CLASSIE requires a more complex approach to equation solution by machine code. The procedure must follow the general flow shown in Figure 4.5. The essence of the factorization on the subcircuit matrices differs from that performed on the interconnection matrix in the type of arithmetic used. As already mentioned, the system matrix is solved most efficiently using the scalar processor whereas the multiple instances of subcircuits suggest a vector solution as most advantageous. The sequence of operations for the solution of a BBDF matrix point to the use of three machine code tables. The first table from which to execute is MADCMP and contains the code for performing LU decomposition and forward substitution on the matrices for all subcircuits. This phase updates also the terms in the interconnection matrix and RHS. Next the system-level equations are

solved executing a scalar LU, forward and back substitution. The machine code is stored in table MACINS. The solution for all variables in the subcircuits is found using the code in the third table, MASOL. The solution of the global variables are broadcast into the solution vectors of the subcircuits where they are external variables. A vectorized back substitution finds the complete solution vector for each cell.

The machine code is generated by routines CODGEN which performs a symbolic scalar Gaussian elimination and SCODGN which generates the instructions for solving the diagonal submatrices of the BBDF. The entire block needed to store each of the instruction tables is allocated only once based on the instruction count computed already during the actual Fortran factorization step. This approach of allocating space minimizes the overhead of generating the machine code because the references to the memory manager and the associated table search and moving of memory blocks are minimum. The scalar solution needs only four operation types.

1. IF (ABS(PIVOT).LT.PIVTOL) PIVOT=PIVTOL
2.  $A = A / B$
3.  $A = A - X \times C$
4.  $A = A - B \times C$

Operation codes 3 and 4 are identical with the only difference that X refers to an operand already available in one of the CPU registers. In the CRAY-1 implementation, the eight primary scalar registers are used in such a way as to achieve most parallelism in execution.

The operations involved in the solution of the diagonal submatrices of the BBD are more numerous because of the diversity of operations between scalars and vectors. The operations are listed according to the data flow

shown in Figure 4.7. In this figure are presented all the operations performed by the code contained in table MADCMP. There are six different operation codes necessary to perform the computations of Figure 4.7.

$$1. \quad V_{pivot} = \begin{cases} v_{diag} & \text{if } v_{diag} > PIVTOL \\ PIVTOL & \text{if } v_{diag} \leq PIVTOL \end{cases}$$

$$2. \quad V = Y / V_{pivot}$$

$$3. \quad V_1 = Y_1 - V \times V_2$$

$$4. \quad S = S - V \times V_2$$

$$5. \quad V = V - Y_1 \times V_2$$

$$6. \quad S = S + V$$

In the above list of operations  $V$  refers to vector registers and  $S$  to scalar registers. It is to be emphasized that vectors are defined across all instances of the same cell definition. The chaining of operations 1. to 4. is achieved following the LU decomposition. Thus Op-code 1 achieves a vector merge between the elements of the diagonal vector  $V_{diag}$  and the scalar PIVTOL. The latter is stored in the locations of the pivot vector  $V_{pivot}$  which correspond to elements  $v_{diag}$  which do not satisfy the absolute minimum criterion

$$v_{diag} > PIVTOL.$$

This process is entirely data dependent and cannot be predicted at the code generation time. The action taken in this situation differs between a Fortran solver and the machine code. In the former case a 'change of pivot on the fly' is performed which consists in a reordering of the remainder matrix at the step where the absolute minimum criterion is violated. In the latter case of machine code execution however, the entire code must be executed

```

for (i=1; i<=Nint - 1; i=i+1) {
  for (j=i+1; j<=N; j=j+1) {
    LOCATE Aij and Aji
    Vfor all instances l (l=1; l=l+1)
      Aji(l) = Aij(l) / Aii(l)
    for (k=i+1; k<=N; k=k+1) {
      LOCATE Aik and Ajk
      Vfor all instances l (l=1; l=l+1)
        Ajk(l) = Ajk(l) - Aji(l) × Aik(l)
      if (j>Nint & k>Nint) {
        for all instances l (l=1; l=l+1) {
          xz = index (l)
          Yxz = Yxz - Aji(l) × Aik(l)
        }
      }
    }
  }
}

for (i=2; i<=N; i=i+1) {
  for (j=1; j<=min(i-1, Nint), j=j+1) {
    LOCATE Bi, Bj, Lij
    Vfor all instances l (l=1, l=l+1)
      Bi(l) = Bi(l) - Bj(l) × Lij(l)
  }
}

for all instances l (l=1, l=l+1) {
  for (i=1, i<=Next, i=i+1)
    LOCATE RHS Bi in subcircuit and Cx in intercon matrix
    Cx = Cx + Bi(l)
}
}

```

Figure 4.7. BBD submatrix LU decomposition, FS and RHS update

until a return jump to the controlling Fortran code is encountered. The merge of the minimum value PIVTOL is necessary for the completion of operations without the occurrence of any floating exceptions. At the time of the return to the driving Fortran routine an error flag is checked which is set by the code in tables MADCMP or MACINS every time a diagonal element is smaller than PIVTOL. If the error flag has been set a reordering of the corresponding submatrices is performed and new machine code is generated.

Once the pivot vector has been found it is scattered back to memory. The next operation at each step is the division of the column elements by the pivot which is preserved in memory in one of the vector registers. Only one vector gather is performed at each operation Type 2. The resulting L vector is preserved for the subsequent operation Type 3. This is a triad operation which needs two vector gather,  $V_1$  and  $V_2$ . For each code type the underlined operands need be gathered (loaded) from memory. The last op-code used in the LU factorization is Type 4. It updates the scalar entries corresponding to each cell instance in the interconnection matrix. The product  $V \times V_2$  has been already computed and is stored in a vector register. A scalar load and subtract is performed for each cell instance.

Op-code 5 represents a triad operation which seems identical to that of Op-code 3. The former is used in the forward substitution step which performs an accumulation for each RHS element traversing the L triangular submatrix row by row. and multiplying each row by the RHS. A start-up code gathers  $V$  which is then kept in that vector register until all vector products are subtracted after which it is scattered back to memory. Thus Op-code 5 is also a triad with vector operands  $V_1$  and  $V_2$  being gathered for each

nonzero in a row of submatrix  $L$ .

To each RHS element for an external node equation of each cell instance corresponds a RHS element of the interconnection matrix. This latter element must be updated after the FS performed on the subcircuit matrices. This operation is achieved by Op-code 6. A start-up code gathers the vector for all cell instances and is followed by a scalar load, add and store of each interconnection RHS element with the appropriate vector element pointed to by a counter.

The LU factorization, FS and BS for the interconnection matrix is performed next by the scalar code of table MACINS. This is followed by the back substitution for the subcircuit matrices. The flow of operations is shown in Figure 4.8. The machine code stored in MASOL needs two new operations beyond the six vector codes defined above. Two new operation codes, 7 and 8, are introduced and code Type 5. can be used again for the triad operation of Figure 4.8. The following three codes are used in this BS phase.

$$7. \quad \mathbf{V}_{sol} = \mathbf{S}_{sol}$$

$$5. \quad \mathbf{V} = \mathbf{V} - \mathbf{Y}_1 \times \mathbf{V}_2$$

$$8. \quad \mathbf{V}_{sol} = \mathbf{V} / \mathbf{Y}_{diag}$$

Op-code 7 assigns element by element to each vector representing all instances of an external node solution the appropriate scalar solution from the system equations. An additional code is used at completion of the loop to scatter the vector elements to the appropriate locations in memory. Op-code 5 and 8 are used for the solution of the internal variables of each subcircuit. The former updates the value of the RHS based on the upper triangular elements of the respective row gathered in  $\mathbf{V}_1$  and the variables already solved for and gathered in  $\mathbf{V}_2$ . The result of this triad is left in

```

for (i=1, i<=Next, i=i+1) {
  for all instances l (l=1, l=l+1) {
    LOCATE solution Xi in subcircuit and Zj
    in interconnection matrix
    j=index(l)
    Xi = Zj
  }
}

for (i=Nint, i>=2, i=i-1) {
  for all instances l (l=1, l=l+1) {
    for (j=i, j<=N, j=j+1) {
      LOCATE Bl, Bj, Uij
      Bl = Bl - Uij × Xj
    }
    Xi = Bl / Uii
  }
}

```

Figure 4.8. Back Substitution for the Subcircuit Matrices

vector register  $V$  and is used by Op-code 8 to perform the division by the diagonal  $V_{diag}$ . It can be noticed that Op-code 8 differs from Op-code 2 in the attribute of the vector already available in a register which is the dividend in the former and the divisor in the latter case.

The new vector code written for CLASSIE takes full advantage of the eight vector registers of the CRAY-1 architecture to minimize memory access, overlap and chain execution in the various functional units. A special case occurs when there are more than 64 instances of the same subcircuit. All the vector operations listed above have to be performed as many times as there are groups of 64. This is taken care by the code generation routine SCODGN which generated also the right table offsets for each vector computation.

## CHAPTER 5

### Performance Evaluation of CLASSIE

#### 5.1. Introduction

This chapter presents an approach to evaluate the performance of a circuit simulation program. An estimate of the CPU time can be made once two key parameters are known about the two major parts of the simulation.

Based on this criterion both SPICEV and CLASSIE are compared with SPICE2. Several benchmarks of medium to large complexity are used in this comparison. A careful analysis of the limitations in the performance gain and a breakdown of the different tasks in the simulation is presented. Although SPICEV has not been a final goal in this project it is a useful program and is also used here for demonstrating the impact of different speedup techniques.

Since CLASSIE has been designed as a general-purpose simulator, it is also checked out by running the standard SPICE2 benchmark set. These are small circuits with less than 50 transistors with run times which are usually less than one second for a transient analysis. The results lead to some interesting conclusions.

#### 5.2. Speed-Performance Evaluation

It is generally accepted that the circuit complexity can be characterized by a generic number 'N' of devices or nodes on the assumption that the

number of semiconductor devices and nodes (equations) of a circuit are approximately equal or at least have a constant ratio. It is also generally accepted that analysis time increases with the power 1.2 - 1.4 of the circuit complexity 'N' [DeMa79]. Circuits analyzed with CLASSIE on a CRAY-1 contradict both of the above generalizations.

A more accurate time model is necessary in order to predict the performance of a circuit simulator. Semiconductor devices are represented in most circuit simulators, such as SPICE2 [Nage75], [Cobe76], [Vlad81b], ADVICE [Nage80], ASPEC [Jenk82], SLIC [Kop75], and CLASSIE, by a variable number of equations depending on whether parasitic terminal resistances are specified or not. For this reason there is an arbitrary relation between the number of devices and that of nodes as brought out in the large circuit examples listed below.

Two parameters have been found to provide a rather accurate characterization of the simulation time. The parameters refer to the two major parts of the analysis: semiconductor-device-model evaluation (Jacobian terms) and linear-equation solution. Parameter  $t_d$  is the time for one model evaluation in one iteration. The second parameter is  $t_e$ , the time for solving one equation in one iteration. The analysis time can then be estimated as:

$$T = n_{iter}(n_d * t_d + n_e * t_e) + \text{overhead} \quad (5.1)$$

where  $n$  represents the number of entities designated by the subscript, i.e.,  $n_{iter}$  is the iteration number,  $n_d$  the number of devices and  $n_e$  the number of equations. From above it is obvious that the speedup one can get in circuit simulation depends on how much the two characteristic times can be reduced. The particular circuit determines the relative weight of the two terms in parentheses. The analysis time for a subcircuit-oriented program

like CLASSIE can be expressed as:

$$T = n_{iter}[n_{di} * t_{di} + n_{ei} * t_{ei} + n_{ss} * (n_{ds} * t_{ds} + n_{es} * t_{es})] + overhead \quad (5.2)$$

where  $n_{ss}$  is the number of subcircuits and the second subscript 'i' stands for interconnection while 's' stands for subcircuit. As shown previously  $t_e$  can be reduced greatly by code generation [Vlad81a], [Coh81], [Vlad82]. In order to reduce  $t_d$ ,  $t_{di}$  and  $t_{ds}$  in the vector processor environment it is obvious that the devices have to be grouped either by subcircuit and/or by model and evaluated in parallel. The vectors are defined in different ways for devices at the subcircuit level and at the interconnection level; the vector length can differ among different subcircuit definitions. Based on this observation the device evaluation times  $t_d$ , for a SPICEV simulation, and  $t_{di}$ , and  $t_{ds}$ , in the case of CLASSIE, are different for the same LSI circuit.

### 5.3. Benchmark Presentation

A number of large circuits containing from a few hundred to over one thousand devices or equations have been analyzed. Two typical circuits have been chosen as examples. Each is built up using cells, either a bipolar NAND gate and or a MOS operational amplifier, together with interconnection circuitry. The size of the circuit is easily varied changing the number of instances of the different cells. In the case of the MOS filters the circuitry at the system level (MOS switches and capacitors) also increases with complexity. A statistical description of the benchmarks is given in Table 5.1. from both the point of view of an element-by-element representation as in SPICE2 and a two-level hierarchical representation as used by CLASSIE.

The three adders are all-NAND circuits containing approximately 60% bipolar transistors and 40% diodes. Figure 5.1 shows the system-cell con-

<b>SPICE2</b>					
<b>Circuit</b>	<b>Dev</b>	<b>Eqs</b>	<b>%Sprs</b>	<b>%Mod.Ev</b>	<b>%Eq.Sol</b>
<b>Adder1</b>	<b>72</b>	<b>118</b>	<b>95.14</b>	<b>44.3 (64.1)</b>	<b>52.2 (8)</b>
<b>Adder4</b>	<b>288</b>	<b>450</b>	<b>99.3</b>	<b>40.7 (69.6)</b>	<b>57.2 (11)</b>
<b>Adder16</b>	<b>1152</b>	<b>1747</b>	<b>99.65</b>	<b>37.3 (83)</b>	<b>61.0 (9.7)</b>
<b>Lowpass</b>	<b>70</b>	<b>42</b>	<b>83.89</b>	<b>73.4 (79.5)</b>	<b>11.9 (1.7)</b>
<b>Filter</b>	<b>756</b>	<b>410</b>	<b>97.6</b>	<b>66.7 (75.6)</b>	<b>23.6 (2.5)</b>
<b>CLASSIE</b>					
<b>Adder1</b>	<b>0</b>	<b>16</b>	<b>64.06</b>	<b>69.7</b>	<b>14.2</b>
<b>Adder4</b>	<b>0</b>	<b>55</b>	<b>87.14</b>	<b>74.5</b>	<b>16.7</b>
<b>Adder16</b>	<b>0</b>	<b>163</b>	<b>94.45</b>	<b>75.7</b>	<b>18.6</b>
<b>Lowpass</b>	<b>20</b>	<b>26</b>	<b>74.7</b>	<b>81.6</b>	<b>6.3</b>
<b>Filter</b>	<b>181</b>	<b>157</b>	<b>93.3</b>	<b>86.9</b>	<b>7.1</b>
<b>Subckt.</b>	<b>Dev</b>	<b>Eqs</b>	<b>%Sprs</b>	<b>Instances</b>	
<b>NAND</b>	<b>8</b>	<b>16</b>	<b>62</b>	<b>9, 36, 144</b>	
<b>OPAMP</b>	<b>25</b>	<b>16</b>	<b>45</b>	<b>2, 23</b>	

Table 5.1. Benchmark Statistics

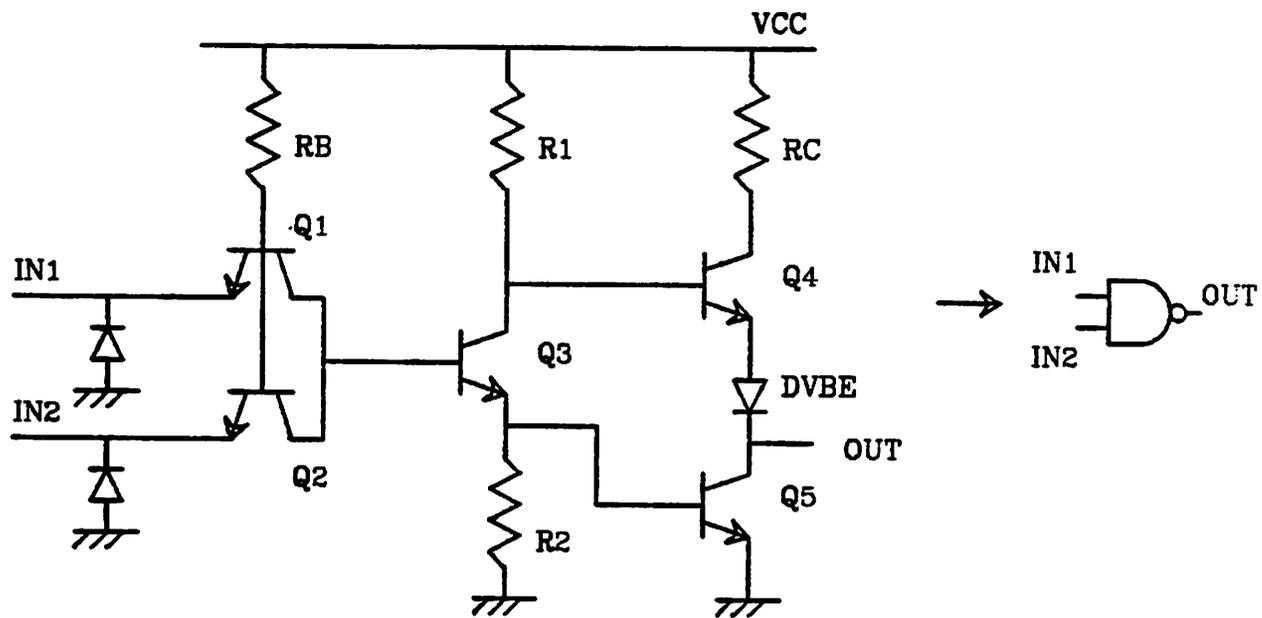
cept for the Adder4 and Figure 5.2 the waveforms which exercise the circuit in a extensive transient analysis. Filter is an NMOS switched-capacitor, lowpass filter containing 10 lowpass sections with two operational amplifiers per section and two antialiasing and reconstruction circuits.

Four equations out of sixteen representing the NAND subcircuit correspond to external nodes. The OPAMP circuit has five external nodes. The external node contributions are gathered in the interconnection matrix. It is interesting to note that two cells of totally different function and complexity, 8 devices for the NAND gate versus 25 for the OPAMP, have the same size matrix representation.

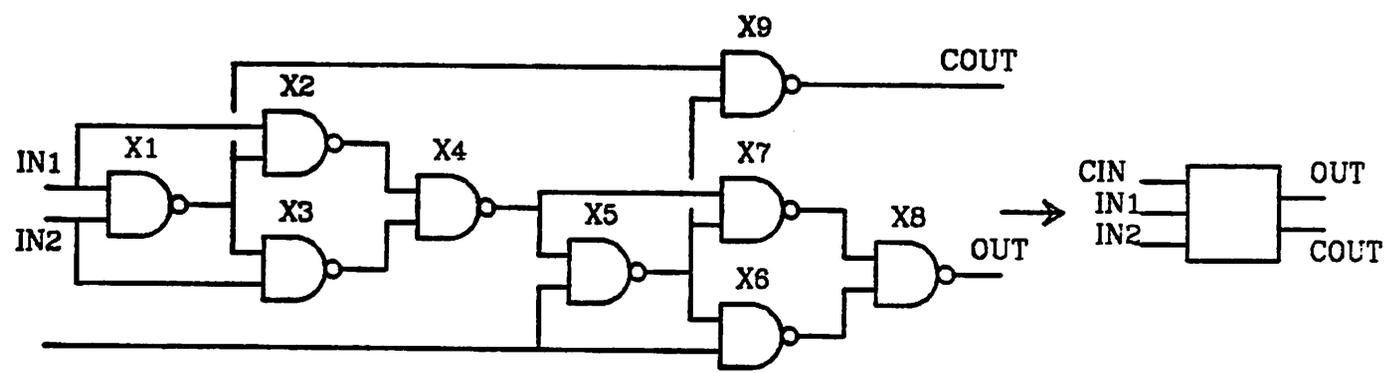
In Table 5.1 can be found also the percentage contributions of model evaluation and linear equation solution to the simulation of the respective circuits. The data written in parentheses for SPICE2 are obtained from runs using scalar-code generation. The increase in relative importance for the Fortran equation solution can be explained by both the increase of search with increasing complexity as well as by the reduction of the model evaluation as more devices are bypassed.

#### **5.4. SPICEV Performance**

The performance of SPICEV is a good place to start the comparison with a standard version of SPICE2 because there is a one-to-one correspondence between the features of the two. Everything is similar in the two programs with the exception of the machine code solver (scalar), the ordering of devices by models and their linearization in vector mode and the vectorization of the truncation-error computation in transient analysis.

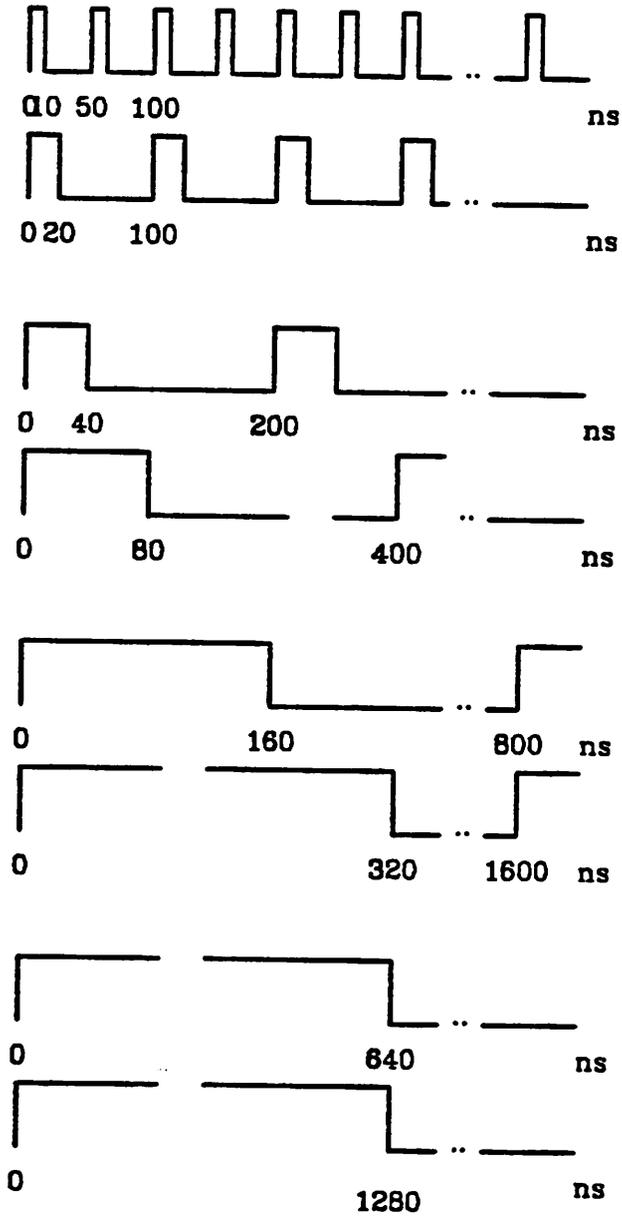


NAND Subcircuit

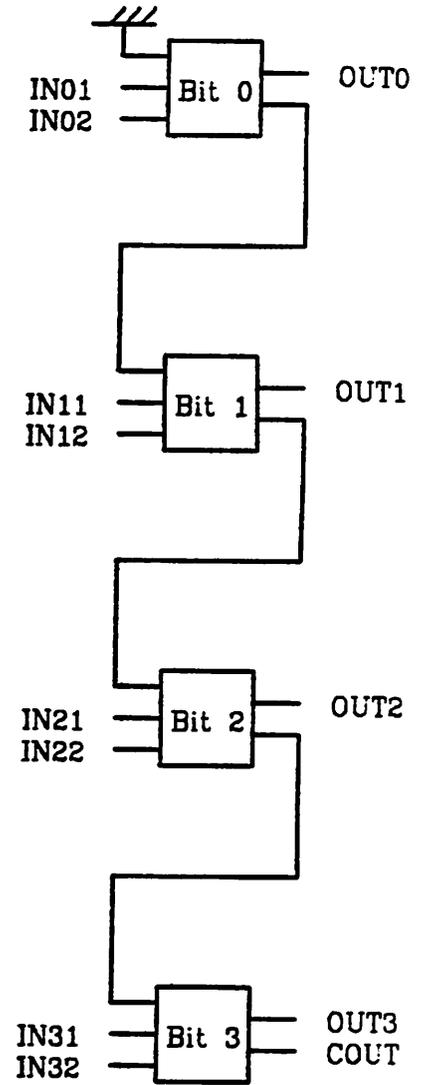


One-Bit Adder

Figure 5.1 NAND Subcircuit and One-Bit Adder (Adder1)



Input Waveforms 0-3200 ns



Four-Bit Adder

Figure 5.2. Four-Bit Adder (Adder4) and Transient Representation

Table 5.2 summarizes the two characteristic times introduced in the first section of this chapter,  $t_d$  and  $t_e$ , as well as the time per iteration and the speedup factor between the two programs. A first observation from the above data is the very large effect of the scalar machine-code generation which cuts the equation solution time by a factor of 8 to 12. Another observation can be made regarding  $t_e$  which increases with the number of equations for the Fortran solver while it differs very little for the different circuits when the solution is performed by machine-code. The difference in  $t_e$  for the first four circuits on one hand and the fifth on the other, is due to the number of floating-point operations which become the dominant factor for a machine-code solution. The ratio between floating-point operations and equations, listed in Table 5.2, proves the above point. There are one third more floating-point operations per equation in average for the Filter which makes  $t_e$  for this circuit to be higher in comparison with the other circuits. There is little difference in the flops/equation ratio for the other four circuits. The effect of this number seems to be lost in other search and memory operations when the Fortran solver is used.

$t_d$  is also a very important number. The model vectorization is seen to bring about a speedup of about 1.5 for the bipolar mix (diode and BJT) and around a factor of 2 for the MOS circuit. The speedup for this part of the simulation is not larger because more computation is performed to evaluate all possible formulations of equivalent conductance which depend on region of operation. Another factor is that the parameter gather and conductance scatter is not vectorized. Data taken on a CRAY-1 simulator [Cala81] show that for a BJT 75% of the total time of conductance evaluation (excluding limiting and truncation error) is spent in gather/scatter when full 64 elements long vectors are used. The larger value of  $t_d$  for smaller

Param	Add1	Add4	Add16	Lowpas	Filter
<b>SPICE2</b>					
$t_d(\mu s)$	51	45	46	121	99
$t_e(\mu s)$	38	41	47	30	47
<b>SPICEV</b>					
$t_d(\mu s)$	28	27	27	56	52
$t_e(\mu s)$	4	4	4	4	6
speedup	2.3	3.2	3.5	2.6	2.4
flops/eq.	12.4	13.4	14.1	14.7	20

**Table 5.2. SPICEV vs. SPICE2 Performance Comparison**

circuits run on SPICE2 (see Table 5.2) is the result of less bypass than for a large circuit.

The contributions of the solution techniques used in SPICEV to the overall simulation speed is depicted in the graphical representation of Figure 5.3. The performance points are measured for five adders increasing in ratios of two from Adder1 to Adder16, simulated by SPICE2 with and without machine code solver and by SPICEV. The complexity of the circuit is given in number of semiconductor devices for this graph. The simulation time in all three cases increases with an exponent of 1.05 with circuit complexity. The three curves are separated by the difference in speed caused by the code solver alone and together with vectorized device evaluation. The linear increase in simulation time expected from the machine code solution is altered by the reordering performed at the beginning of transient analysis. The impact of the reordering is however diminished for this transient simulation compared to the DC operating point analysis presented in Chapter 3, due to a two-order of magnitude larger number of iterations.

The speedup factor is influenced by several elements such as the ratio of  $t_d$  vs.  $t_e$  and of  $n_d$  vs.  $n_e$ . The speedup is larger for the bipolar circuits because the contribution of equation solution in all-Fortran SPICE2 (see Table 5.1) is much larger than for the MOS circuits. The large gain in speed due to the machine-code solver contributes already a factor of two for the bipolar circuits. This part is reduced more effectively by the code solver than the model evaluation is by vector operations.

This difference can be looked at also from the perspective of the device-equation ratio which is roughly a factor of two for the MOS circuits and 0.6 for the bipolar circuits. For the MOS circuits  $t_d$  is the double of  $t_e$  to

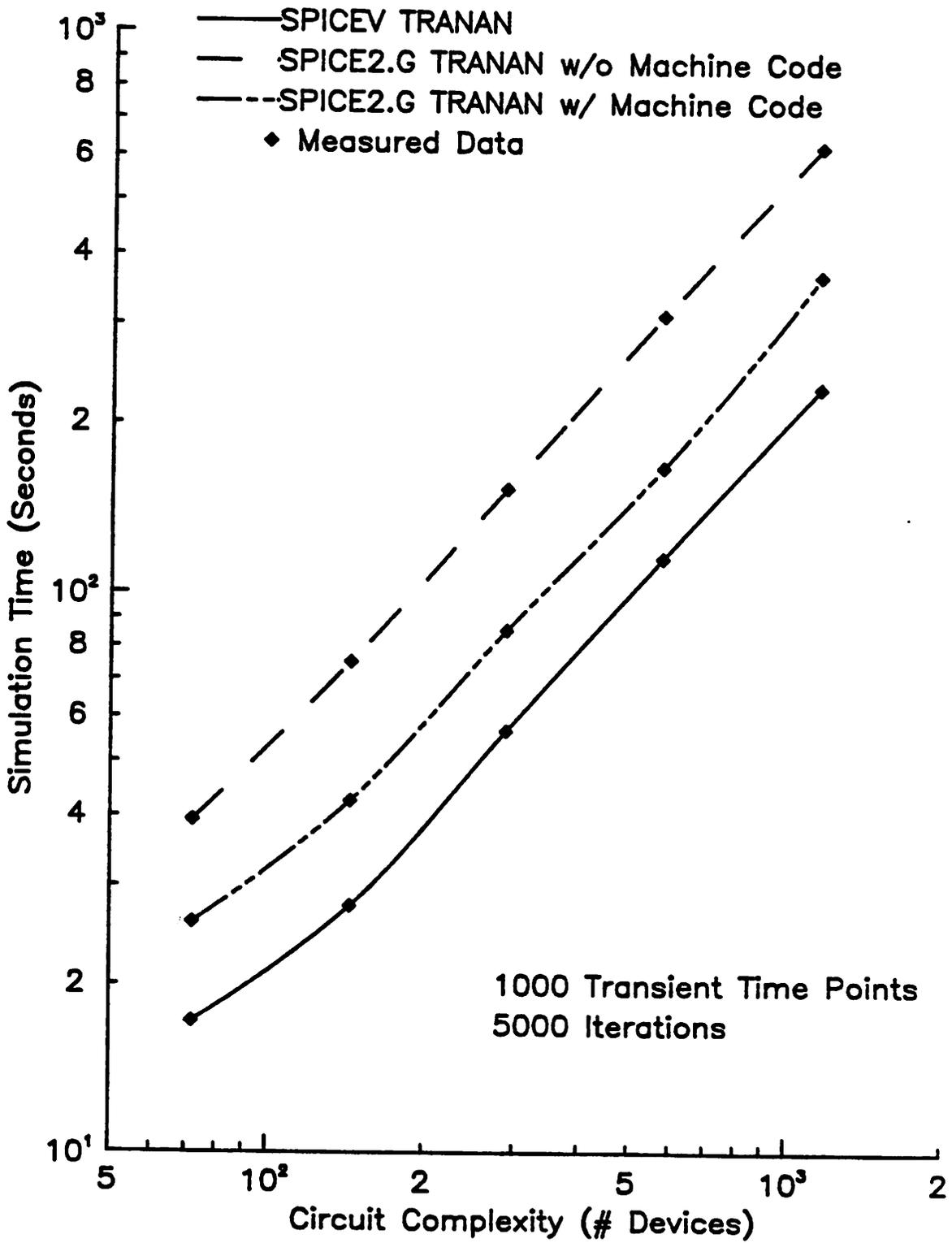


Figure 5.3 SPICEV/SPICE2 Comparison for Adder Circuits

start with and can not be reduced by more than a factor of two. In SPICEV  $t_e$  is approximately the same for all circuits but  $t_d$  is twice as large for MOS-FETs compared to the diode-BJT combination. All these considerations explain the difference in speedup between the two types of circuits.

### 5.5. CLASSIE Performance

The presentation of the results obtained from runs of CLASSIE is displayed in Table 5.3. Two sample CLASSIE outputs for the Adder4 and Adder16 are listed in Appendix 4. Again extensive use is made of the characteristic times introduced in Equation 5.2. The last column of the table lists the speedup compared to SPICE2. As already mentioned the characteristic times differ between the interconnection circuitry and the subcircuits because of the gather/scatter which is part of  $t_{dx}$  where x stands for either i for the interconnection and s for the subcircuits, respectively.  $t_{ds}$  differs also for various subcircuit definitions depending on how many cells reference each definition; the number of instances defines the vector length used in the model evaluation and thus influences  $t_{ds}$ . The equation-solver characteristics are also different based on the use of vector code for the subcircuits and scalar for the interconnection. Both  $t_{ei}$  and  $t_{es}$  vary among circuits and subcircuits, respectively, depending on sparsity patterns and number of operations as has been shown also for SPICEV.

The data in Table 5.3 should be viewed in conjunction with the circuit statistics presented in Table 4.1 and 5.1. An important specification is that the circuits for the runs for this table have just one parasitic resistance in the BJT model and none in the MOSFET model. The cells, NAND and OPAMP, are only components of the large circuits and therefore have no entries in

<b>CLASSIE</b>	$t_{dx}(\mu s)$	$t_{ox}(\mu s)$	<b>MFLOPS</b>	<b>Speedup</b>
Adder1	-	4.4	5.1	4 / 4.3 / 4.2
NAND	19	1.5	13.8	-
Adder4	-	5.4	5.3	5.2 / 10 / 5.5
NAND	18	1.2	17.6	-
Adder16	-	6.7	5.4	5.7 / - / 6.9
NAND	16	1.2	18	-
Lowpass	56	5	5.3	2.6 / 2.9
OPAMP	56	13.6	3.3	-
Filter	52	6.6	5.2	3.7 / 5
OPAMP	29	3	14.5	-

Table 5.3. CLASSIE Performance

the 'Speedup' column since no comparison can be made independently from the overall circuit. All semiconductor devices of the adder circuits are introduced as part of the NAND cells; this fact explains the absence of any entry in the  $t_{dx}$  column. Any other blank entry is caused by the absence of the corresponding data point.

The reduction in characteristic times for the subcircuit can be seen to be larger with increasing number of instances. From the analysis of the results obtained with SPICEV it is expected that the speedup is larger for the adder circuits compared to the filters. A first observation relates to  $t_{ds}$  for MOSFETs which is reduced by another factor of almost two compared to the time in SPICEV. The devices at the interconnection circuitry, 181 out of 756 MOSFETs, are still characterized by a  $t_{di}$  of 52  $\mu$ s. The corresponding reduction in the same parameter for the bipolar mix is more like 25 to 30%.  $t_{es}$  for the Lowpas section is much larger than the corresponding time for the Filter because of a vector length of only two in the linear equation.  $t_{ds}$  is not as much affected because of the long vectors used in model evaluation.

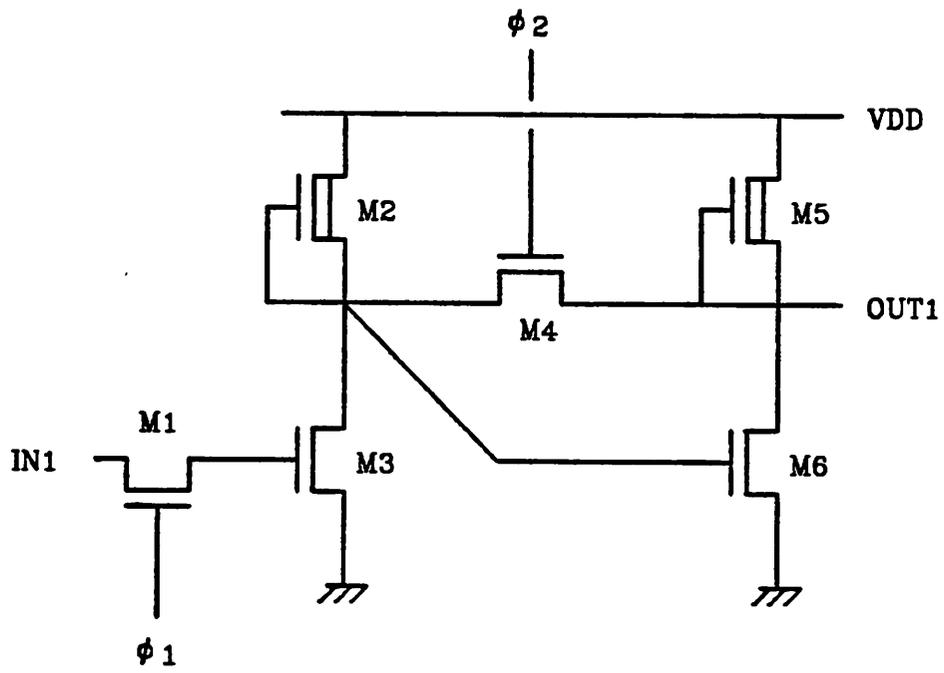
Two numbers characterize the sparse solver; one is the parameter  $t_{ex}$  while the second is the Megaflops rate. The Mflops rate is a better characteristic of the solver than the  $t_e$  parameter. The reason for it is that the operation count provides the best measure of the computational effort. This number proves to be stable for different sparsity patterns and is therefore a good characteristic of the sparse solver on the CRAY-1. The Mflops rate is computed from the run statistics time for the subcircuit solver, the total number of iterations, the number of operations for each subcircuit matrix and the number of instances:

$$\text{Mflops rate}_{\text{vcode}} = \frac{n_{\text{iter}} \cdot n_{\text{ss}} \cdot n_{\text{ops}}}{T_{\text{vcode}}} \quad (5.3)$$

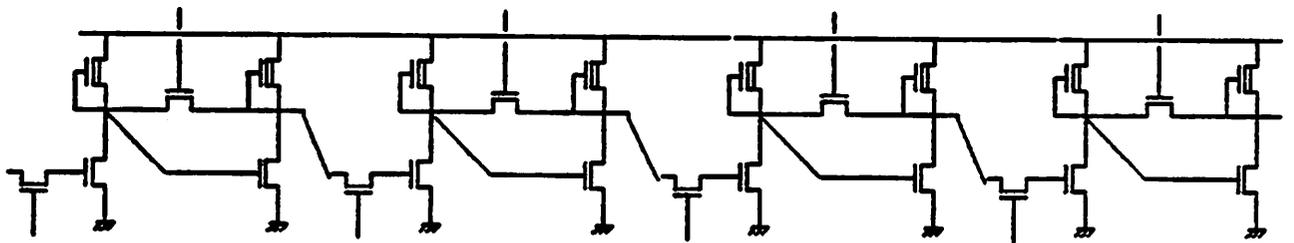
The same equation is used for the Mflops rate in the interconnection matrix with the product  $n_{\text{ss}} \cdot n_{\text{ops}}$  replaced by the total number of operations for this matrix, number which is printed in the accounting information. The computational effort of a Gaussian elimination has been measured in number of multiplications and divisions; this number is printed by SPICE2. The time for add and subtract has been neglected because historically scalar computers have always had a much larger execution time for multiply and divide. On the CRAY-1 all four basic arithmetic operations are characterized by execution times of the same order, 6 cp cycles for addition and division, 7 for multiplication and 14 for reciprocal approximation. In SPICE2, SPICEV and CLASSIE all the floating-point operations are counted when they execute on the CRAY-1. Both for SPICEV and CLASSIE (interconnection equations) the scalar solver performs at 5.3 Mflops.

The vector solver is more dependent on the matrix structure and vector length (instances). The speed is between 13.8 - 18 Mflops which is impressive but is below that predicted by Calahan [Cala80]. As in the case of SPICEV the speedup for the MOSFET circuit is lower, 3.7, compared to 5.2 in the case of the Adder4 and 5.7 for the Adder16. Changing the mix between equations and devices by introducing parasitic series resistances in the models brings about higher speedups as predicted by Equation 5.2. The speedup of 10 for the Adder4 with two parasitic resistances is due to an additional reordering of the circuit equations performed by SPICE2 which increases considerably the number of fill-ins compared to CLASSIE.

From the data in Table 5.3 it can be concluded that independently of the device to equation ratio, the speedup of CLASSIE increases with the size



One-bit Cell



Four-bit Cell

Figure 5.4 Two Subcircuit Definitions for the Shift Register Circuits

of the circuit. The speed increase with complexity is larger for circuits with a higher percentage of equations as can be noticed from the data in the 'Speedup' column of Table 5.3. A possible explanation for the above difference between CLASSIE and SPICE2 is that the simulation time varies differently with circuit complexity for the two programs. A rigorous way of comparing the simulation time of circuits with increasing complexity is to find a trend curve for circuits built from an increasing number of the same cell. Based on the bipolar NAND cell five circuits of increasing complexity, an one-bit, a two-bit, a four-bit, an eight-bit and a sixteen-bit adder can be constructed. A second example is an MOS shift register built from the one-bit latch shown in Figure 5.4. Seven circuits ranging from a four-bit to a 256-bit shift register, in ratios of 2, cover a circuit complexity from 24 to 1536 transistors. More statistical data on the seven shift registers can be found in Table 5.4. The transistor models do not contain any parasitic series drain and source resistances and therefore the number of equations is approximately half the number of devices. The simulation time for 1000 transient-time points (assuming 5 iterations/time point) for the five adders and seven shift registers is plotted in Figure 5.5 as a function of the circuit complexity measured in semiconductor devices. The slopes of the curves for SPICE2 and CLASSIE are almost parallel and very close to a slope of 1. The overall simulation time thus increases with an exponent a little less than 1 for CLASSIE and with an exponent of 1.05 to 1.1 for SPICE2. The equation solution time using Fortran is however expected to increase faster with circuit complexity in order to explain a number of results of this chapter and Chapter 3. The graphical representation of Figure 5.6 plots the equation solution time as a function of the number of equations. The above assumption is checked by the plot which shows an exponent of 1.3 for the Fortran

Circuit	SPICE2		CLASSIE		CLASSIE / SPICE2
	Dev	Eqs	Dev	Eqs	Speedup
Shiftr4	24	21	0	13/-	2/-
Shiftr8	48	33	0	17/11	2.6/2.1
Shiftr16	96	57	0	25/13	3.4/3
Shiftr32	192	105	0	41/17	3.9/3.9
Shiftr64	384	201	0	73/25	4/4.2
Shiftr128	768	393	0	137/41	4.1/4.5
Shiftr256	1536	777	0	265/73	4.4/5

CLASSIE Subcircuits			
Subckt	Dev	Eqs	Instances
LATCH1	6	8	4, 8, 16, 32, 64, 128, 256
LATCH4	24	17	1, 2, 4, 8, 16, 32, 64

Table 5.4. Shift Register Statistics

solver. The time for the Fortran solution is independent of circuit structure as the curves for shift registers and adders overlap. The vector and scalar code solvers depend on vector length, number of internal equations and number of floating-point operations. This dependency explains the difference in the two code solver curves for the two circuit types.

A last important observation concerning the performance of CLASSIE is the choice of the cells (subcircuits). Figure 5.4 shows two possible cells for defining the shift registers described by Table 5.4. The speedup compared to SPICE2 is higher for an equal number of instances when four cells define the subcircuit. The explanation resides in the number of internal equations or equivalently the size of the diagonal submatrices of the BBDF matrix. When the LATCH1 cell is used as building block the size of each submatrix is only two rows and columns, and the advantage of the vectorized equation solution is limited. The LATCH4 cell leads to a submatrix with dimensions of 11 by 11. The dimensions of the interconnection matrix are also different for the two cases and both numbers are specified in Table 5.4. The first number always refers to the circuit built of LATCH1 cells. Figure 5.7 shows the variation of the speedup for the seven shift registers depending on the choice of the subcircuit. Due to the reduced number of instances of the LATCH4 cell in the 8-bit, 16-bit and 32-bit shift registers, the simulation is faster using the LATCH1 cell as building block. For the larger shift registers the larger submatrix size becomes more important. Another important observation is the lower slope of the execution time increase for a small number of devices. This peculiarity of the CLASSIE simulation time characteristic can be explained by the vector efficiency curve of Chapter 2.

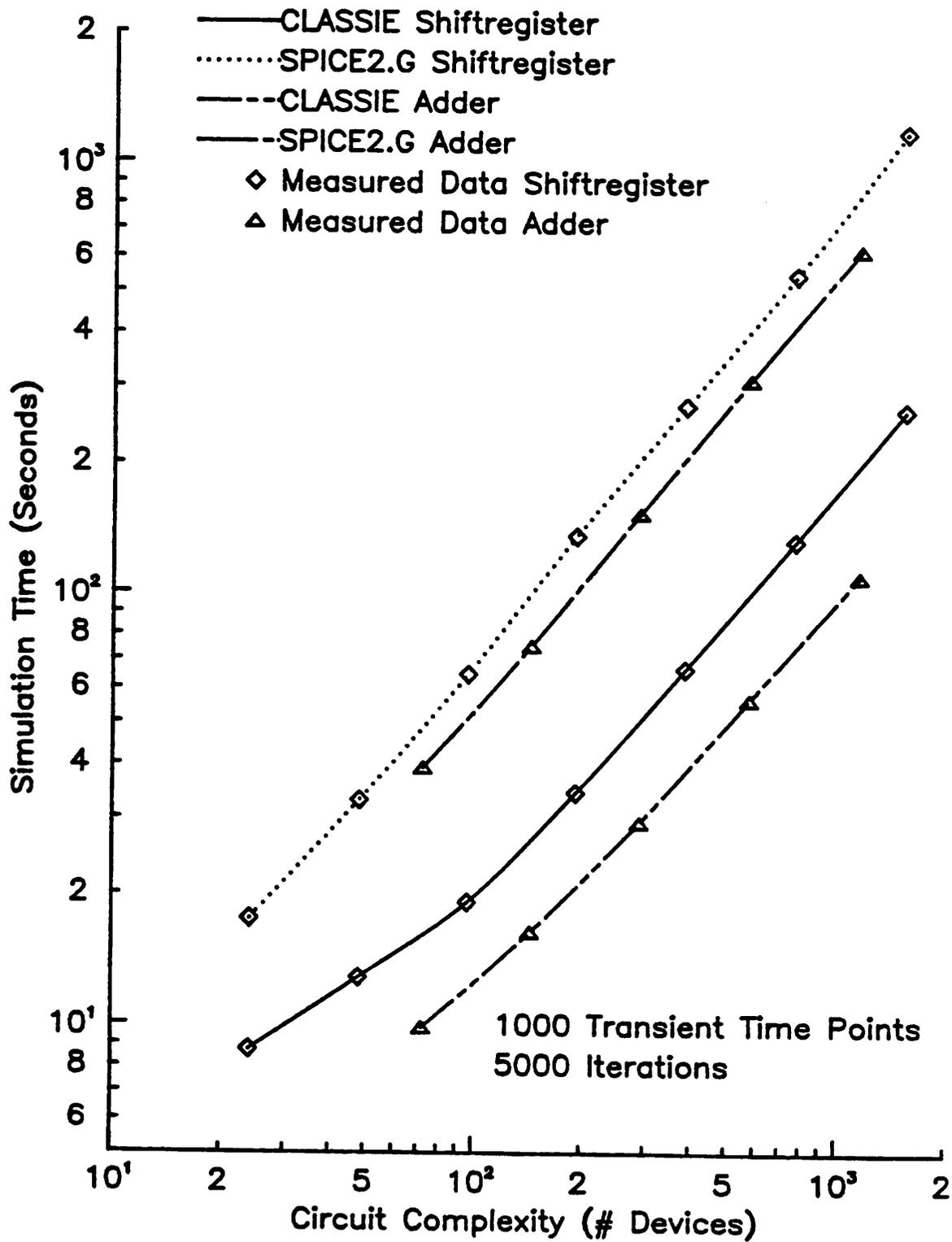


Figure 5.5 CLASSIE/SPICE2 Simulation Time vs. Circuit Complexity

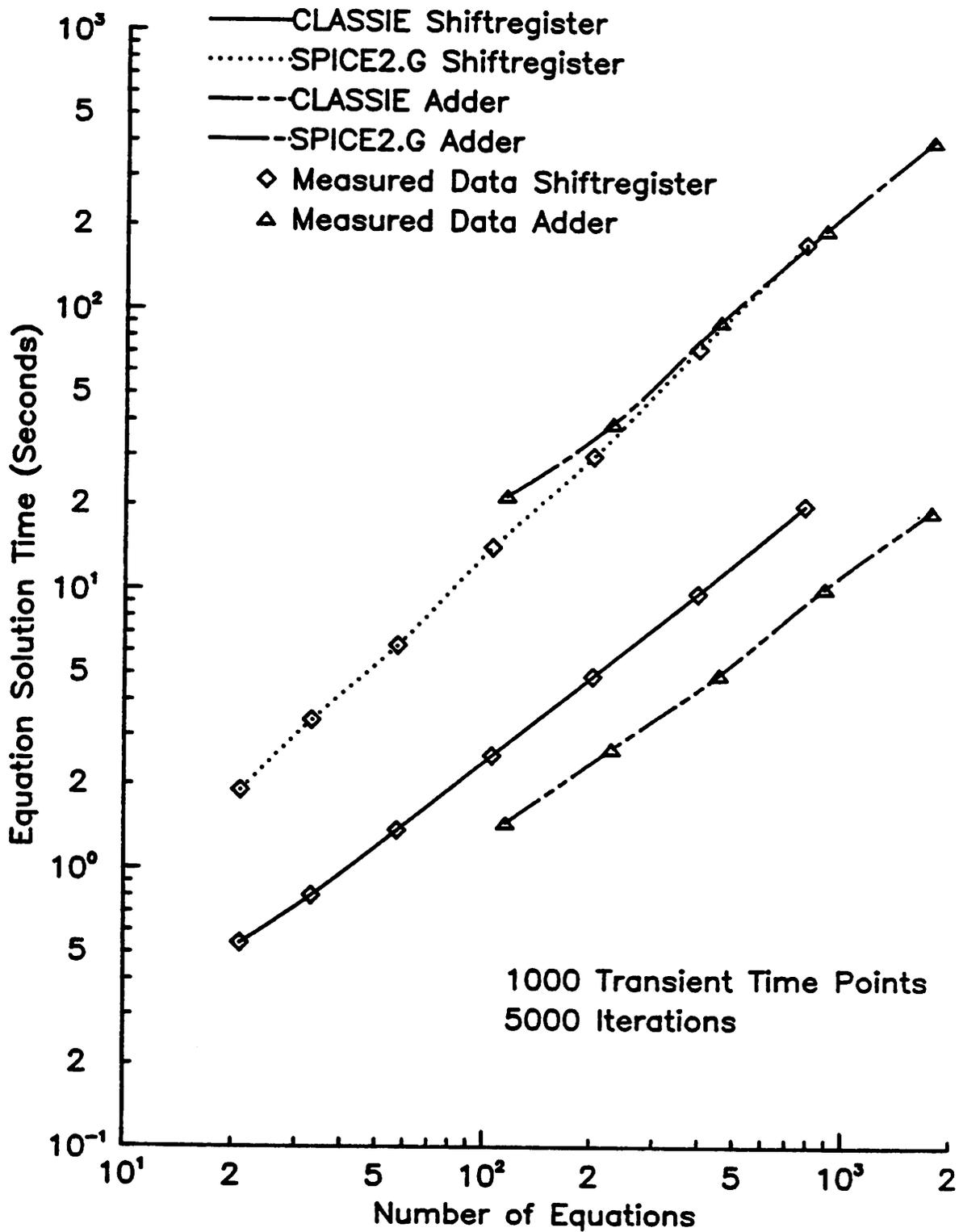


Figure 5.6 Equation Solution Time vs. Equation Number

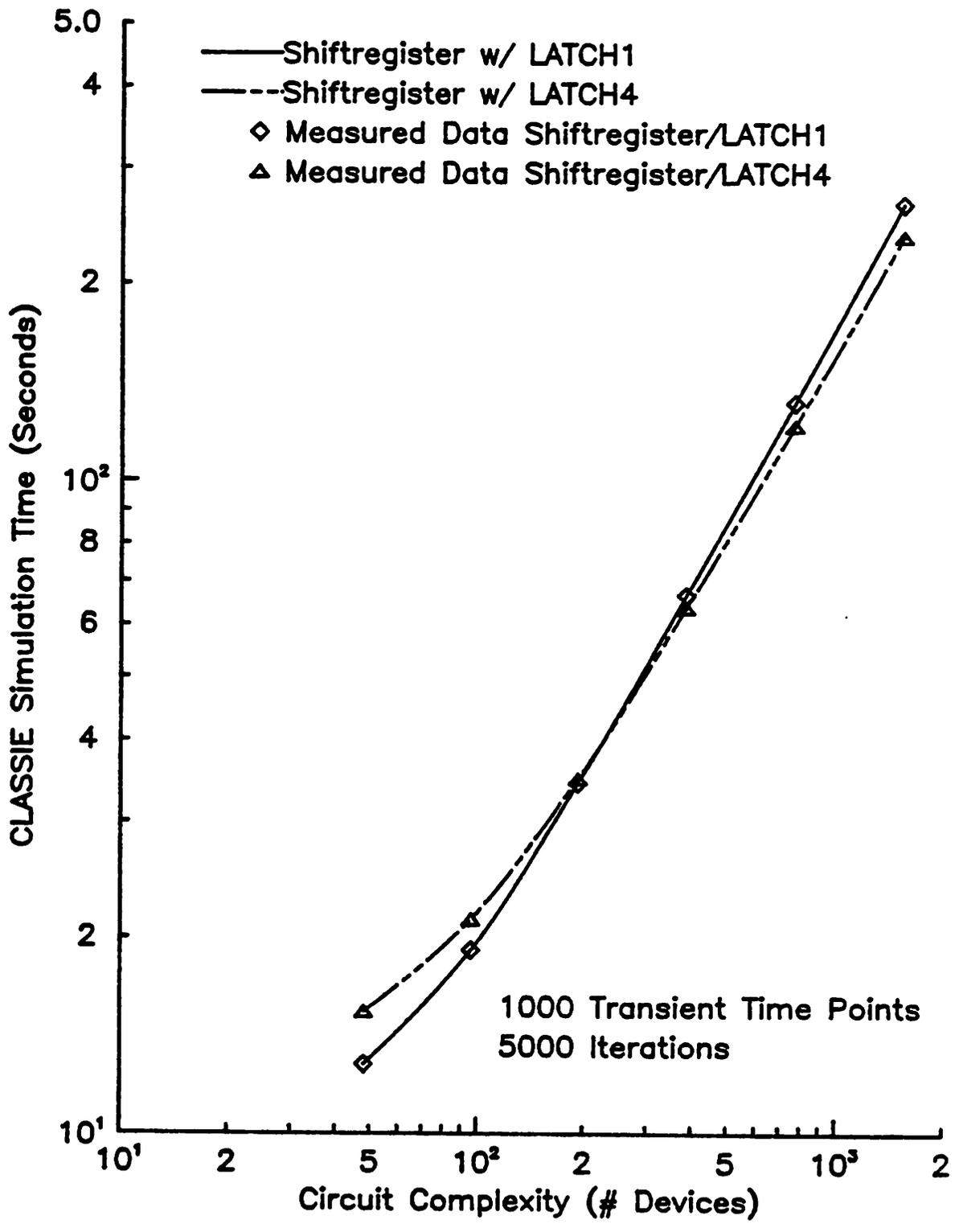


Figure 5.7 Performance Comparison for Two Subcircuits

### 5.6. Simulation Time Breakdown

A most useful tool in improving the performance of a program is the 'Flowtrace' option of the CRAY Fortran compiler, CFT. This option gives a complete image of the average time spent in a certain routine, the number of times it is called, and the percentage contribution to the total CPU run time. In Table 5.5 the results of the choice of the data structures, the two-level analysis and other features of CLASSIE are compared with SPICEV, task by task. From the above data it can be seen that the routines which perform gathering of parameters, initialization, and scattering of matrix terms from and to memory take more than one third of the total time in SPICEV because of the sequential mode in which they are executed. In the percentage for the above tasks is included also the contribution of the decision-making protocol based on analysis status. In CLASSIE the analytical model evaluation has become dominant as desired. Unless a simpler approach to nonlinear semiconductor modeling is used no further speed improvement seems possible. For SPICEV the percentage of the model evaluation is approximately 20% for bipolar and 23% for MOS circuits. This represents in itself an impressive performance of the computational speed of the mathematical functions (exponentiation, logarithm, square root) and floating point units of the CRAY-1. It also explains the smaller percentage of the run time spent by SPICE2 for this part of the analysis on the CRAY-1 compared with other computers.

A major problem for CLASSIE can be a reduced number of occurrences of identically structured subcircuits which increases the time for computation. The approach used to overcome this situation is to incorporate more intelligence into the program; this allows for definition of long vectors in the

Task	BJT Adder4		MOSFET Filter	
	CLASSIE (%)	SPICEV (%)	CLASSIE (%)	SPICEV (%)
Gather/Scatter	16	37.5	29	32.5
Device Eqs.	46	13	17.3	9
D-S Junction Eq.	-	-	11.3	6.5
MOS Capacitance	-	-	5	2.5
Bipolar Limiting	3	7	1	2
FET Limiting	-	-	4	2.5
Eq. Solution	16.7	16	6	5
Convergence Test	2	7	1	1
Subckt. Linkage	7.5	-	2.5	-

Table 5.5. CPU Time Breakdown Comparison

computation part and regrouping according to subcircuits in the gather/scatter part. This approach saves a few percent for large circuits and prevents the deterioration of the computation performance caused by shorter vectors for circuits with fewer instances of a subcircuit. This improvement has brought an additional speedup of up to 10% for the Filter circuit. The Lowpass section is simulated at half the speed of SPICEV without the above approach because a vector length of only two is achieved in vector operations due to only two instances of the subcircuit OPAMP. The use of long vectors in model evaluation however reduces the run time on CLASSIE to that on SPICEV. This result in itself is very important because in the worst case of only 2 instances of a cell the two-level analysis of CLASSIE is equally fast to a vectorized SPICE. The performance of CLASSIE is expected to be superior for more than two instances of each cell type.

Another important observation is the overhead time. From simulation runs it can be noticed that as the main parts of the analysis (device evaluation and equation solution) are made faster the importance of the overhead grows. The major part of the overhead is contributed in SPICEV by memory-manager operations (moving blocks around), truncation-error computation and convergence checking.

In CLASSIE an important percentage of the time is spent to link the subcircuits with the interconnection matrix and is of the order of 5-10% percent. Another source of concern are the time-step control computations. The adders use iteration-count while the filters use truncation-error time-step-control. During the development of SPICEV the contribution of truncation-error computation has been as high as 15% after the vectorization of the model equations. After the LTE computation has been vectorized

this percentage has gone down considerably. Only 2% of the simulation time is spent in CLASSIE for the truncation-error evaluation. The convergence checking which is performed both on the currents of semiconductor devices and node voltages of the circuit has also been reduced in CLASSIE compared to SPICEV using vector comparison and fast vector accumulation routines as presented in the previous chapter.

The use of table models for devices can reduce by up to 5-10% the overall 30-40% which the evaluation time of analytic equations contributes to the analysis time of CLASSIE. The major advantage of table models in this context is that the same sequence of operations is performed for all devices regardless of the operating region. Table models in circuit simulation have been used for the static characteristic only, viz., [Newt77], [Tana80], [Shim82]. The charges associated with each device are still computed analytically. For a Mflops machine the use of table models limited to the dc characteristic is of secondary importance for speed.

The effect on simulation speed of table models can be approximated by a run of the Filter benchmark using the simple Shichman-Hodges model for MOSFETs. Because this model is so simple, its use provides a good estimate of table-lookup for dc characteristics. The percentage time spent in the 'Device Equation' part, see Table 5.4, which accounts for the computation of the conductances associated only with the modeling of the transport in the inversion channel, is reduced to 3% from 17.3%. This simple routine performs all computation as part of a single vectorized Fortran DO loop and achieves approximately 60 Mflops. The total execution time of CLASSIE can be reduced by a maximum of 30% if it is assumed that all the analytical computation is eliminated through some technique of table look-up.

	SPICE2/SPICEV		CLASSIE		
Circuit	Tables	Code	Tables	V. Code	Sc. Code
Adder4	52020	15686	36855	3961	2260
Adder16	202607	63288	126598	40389	8284
Filter	161019	21038	127980	7501	8019

Table 5.6. Memory Requirement Comparison

For large circuits another significant characteristic is the memory requirement of the simulation. The memory space available for data storage on a CRAY-1 with 1 Mword of installed memory is almost 0.9 Mword. Table 5.6 shows a comparison between the memory requirements of the three largest circuits, Adder4, Adder16 and Filter. The usage of storage space is shared by element tables, matrix information, state vectors and machine instructions for the solution of the linear equations. The storage requirement is reduced by 25-35% in CLASSIE because of the savings in integer tables, both for elements and matrix statistics. It is assumed that the tables containing the output variables are stored on disk. The data in the 'Code' column for SPICE represent the number of words needed for the scalar machine instructions for solving a large matrix. For CLASSIE this memory requirement is divided between the vector code for the subcircuits and the scalar code for the interconnection matrix.

### 5.7. Small-Circuit Performance

CLASSIE, designed as a general-purpose circuit simulator for a vector computer (SIMD architecture), is able to run any of the SPICE2 benchmarks. These circuits have a small number of semiconductor devices, from a few up to 50, and can represent simple cells for a large circuit. The analysis time for most circuits in this class is dominated by the device model evaluation. The linear equation solution part can use up to one half of the analysis time for small bipolar circuits with parasitic terminal resistances, as can be seen in Table 3.3 for the UA741 operational amplifier. The performance data presented in this section, highlight the vector-model evaluation used in CLASSIE in comparison to the sequential evaluation with bypass capability of SPICE2. The impact of the scalar-code solver is not as important for these

circuits because a typical transient analysis needs only one to two hundred time points (a few hundred iterations) which is not long enough time to absorb the overhead time for code generation.

Table 5.7 lists a number of bipolar benchmarks which can be divided into analog circuits, the operational amplifiers UA709, UA727, and UA741, and logic circuits, TTL NAND gates and ECL gates. These circuits have been used since the development of SPICE2 [Nage75], as a standard set of benchmarks for this program. The statistics of transient analyses are presented in this characterization. The output of the CLASSIE runs for the above bipolar circuits and the MOS small circuits, described in Table 5.8, can be found in Appendix 5. Sinusoidal low-amplitude signals are specified as input sources for the analog circuits and input signals switching between logic '0' and '1' are used for the digital gates. The circuits are described in terms of number of devices, number of models, number of devices for each model which defines the vector length and number of equations. The number of iterations and run times are presented for the two programs. It is to be noted that the only algorithmic difference between the two programs impacting small circuits is the Jacobian computation of individual devices which can be bypassed in SPICE2 if the terminal variables have not changed from iteration to iteration at one time point. The speedup for the analog circuits is between 2.5 and 3 whereas for the digital circuits the speedup is only 1.1 to 2. This difference is principally caused by the amount of bypassing which is obviously more important for the digital circuits due to the shape of waveforms. Another reason for this difference is the larger number of transistors in the operational amplifiers and hence the larger vector length. The TTL9200 circuit can be considered as a worst-case test for CLASSIE because of the total of only 8 devices and 4 different models leading

Run Statistics					
	SPICE2		CLASSIE		
Circuit	#Iter	Time(ms)	#Iter	Time(ms)	Speedup
UA741	178	680	166	260	2.6
UA727	178	770	176	290	2.7
UA709	168	680	168	230	3
TTLINV	527	670	458	330	2
TTL74S	455	740	451	540	1.4
TTL9200	440	610	466	540	1.1
ECLGATE	375	660	382	340	1.9

Circuit Statistics				
Circuit	#Eqs.	#Xtor	#Diode	#Device/Model
UA741	52	22	0	16 NPN, 6 PNP
UA727	62	22	0	20 NPN, 2 PNP
UA709	44	15	0	13 NPN, 2 PNP
TTLINV	29	5	3	5 BJT, 3 DIOD
TTL74S	34	7	2	3/2/2 BJT, 2 DIOD
TTL9200	31	6	2	2/2/2 BJT, 2 DIOD
ECLGATE	39	8	2	8 NPN, 2 DIOD

Table 5.7. Small Bipolar Benchmarks on CLASSIE

to a vector length of 2 in each model evaluation pass. It is remarkable that even with a slightly larger number of iterations the penalty is not heavier. The iteration count is within 10% for the two programs with the exception of the TTLINV circuit which benefits in CLASSIE from the updated Jacobian computation at each iteration. This last example points out one of the dangers of defining latency (bypassing) at the level of the output variables: it can lead to accumulation of errors if the error criterion is not appropriate for the example.

A final observation on the data of Table 5.7 is the larger number of iterations per transient time-point for the digital circuits compared to the analog circuits. Both cases use truncation-error time-step control which assumes the continuity of the state variable and its derivatives. This hypothesis is true for a sinusoidal function but not so for trapezoidally shaped pulses. The same logic circuits use roughly 10% percent less iterations when iteration-count time-step control is used. This observation together with assertions of Section 2.5 on dc convergence suggest a new user option, DIGITAL and ANALOG. A different set of initial operating points and time-step control are used depending on the type of circuit.

Table 5.8 lists the MOS SPICE2 benchmarks along with run statistics for the two programs. These circuits cover a number of classes of MOS circuits: NMOS all enhancement, NMOS enhancement-depletion, and CMOS circuits, static and dynamic, digital and analog.

A first general observation is that uniformly CLASSIE runs twice as fast than SPICE2 for circuits which have more than eight devices of the same type. Since the contribution of equation solution to the total analysis time is small for MOS circuits most speedup can be attributed to vectorization in

Run Statistics					
	SPICE2		CLASSIE		
Circuit	#Iter	Time(ms)	#Iter	Time(ms)	Speedup
SATINV	145	90	145	90	1
CMOSNOR	279	260	277	290	0.9
CMOSINV	434	370	400	390	0.9
INVCHN	227	430	215	200	2.2
RATLOG	359	480	358	300	1.6
BOOTINV	143	170	143	120	1.4
MOSMEM	293	560	237	240	2.3
MOSAMP2	270	1410	269	480	2.9
CMOSCKT	171	940	171	470	2
DECODER	833	6080	833	2400	2.5

Circuit Statistics			
Circuit	#Eqs.	#Xtor	#Device/Model
SATINV	8	2	2 NMOS
CMOSNOR	9	4	2 NMOS, 2 PMOS
CMOSINV	7	4	2 NMOS, 2 PMOS
INVCHN	12	10	10 NMOS
RATLOG	15	6	6 NMOS
BOOTINV	10	5	5 NMOS
MOSMEM	14	12	12 NMOS
MOSAMP2	25	27	27 NMOS
CMOSCKT	68	22	10/1 NMOS, 10/1 PMOS
DECODER	36	48	31 EMOS, 17 DMOS

Table 5.8. Small MOS Benchmarks on CLASSIE

the model evaluation and truncation error computation. The only exception is the CMOSCKT which gets a good part of its speedup from the equation solution. All models for this example include parasitic series drain and source resistances. For a few examples CLASSIE uses less iterations than SPICE2 because the former program does not bypass any Jacobian entry computation.

Two very small CMOS examples have been added with 2 PMOS and 2 NMOS transistors each to demonstrate the effect of the overhead of vector operations. The result is that CLASSIE is very little slower than SPICE2. This means that for a minimum number of three devices the vectorized computation of the analytical expressions equals or becomes superior to the scalar computation.

The results in this section do not advocate the use of a CRAY-1 for the characterization of a two-transistor circuit but have been judged very important for new conclusions on the effect of parallel computation in circuit simulation. The qualitative aspect of these comments are useful for a circuit simulator running on a computer with an attached array processor, be it a VAX 11/780 with an attached FPS-164 or a 32-bit microprocessor with an attached Floating-Point Processing Unit (FPU) chip.

## CHAPTER 6

### CONCLUSION

The simulator CLASSIE presented in this dissertation is a prototype program capable of performing very fast, accurate analysis of LSI circuits at the transistor level of modeling. The characteristics of LSI circuits have been combined with the features of vector computers in the design for speed and accuracy of the new prototype. The speed of CLASSIE on a CRAY-1 computer makes possible the economical simulation of entire building blocks of a VLSI chip.

An important goal achieved by this project is the creation of the framework for hierarchical circuit simulation. At the present time a set of algorithms similar to the ones used in SPICE2 are implemented in this two-level simulator. The addition of hierarchy has modified the flow of operations in the solution algorithm but has left intact the accuracy of results and convergence properties of SPICE2. The introduction of decoupling at the cell level creates multiple possibilities for implementing different algorithms, e.g., the Gauss-Jordan (or Gauss-Seidel) Waveform Relaxation algorithms in future work.

The features of the SIMD architecture have greatly influenced the decisions on data structures, circuit and subcircuit element ordering and flow of operations. The present design of the simulator has been aimed for optimal performance on a CRAY-1 computer, but the simulator can be adapted to

other existing vector and array processors, and even to scalar computers.

For the analysis of a large circuit, CLASSIE on a vector computer rates in simulation speed between SPICE2 and a timing simulator. Some of the design considerations of this project may prove useful also for future developments of the SPICE2 program.

### **6.1. Hierarchical Simulation**

Based on hierarchical tearing CLASSIE breaks the analysis of a large circuit into separate entities. Provisions have been made in the data structures to allow for the independent processing of each cell or each category of cells (by cell type). The tearing is achieved at the nonlinear equation level.

The ordering by cell type is taken advantage of in the context of the CRAY-1 by defining vector operations across identically structured cells. This approach has been found to be optimal for this particular processor. The relatively short start-up time for vector operations and the lack of a fast gather/scatter capability have had a decisive influence on the decisions made during the design process of the simulator for a CRAY-1.

Latency is believed to have the biggest potential for savings in simulation time for LSI circuits. However identification of latency requires a large amount of computations. The charges of all semiconductor devices in each cell must be known before the latter can be declared latent. The gather operation at each time point of only the active cells combined with shorter vector lengths due to fewer instances of active cells, increase the overhead considerably. For a different SIMD machine a different approach could be more beneficial.

Latency could be efficiently used on a data flow MIMD architecture. For this architecture only the active subcircuits at each time-point are evaluated on different processor nodes.

An important aspect in CLASSIE is the separation of the analysis at two levels. The data structures and sparse-matrix pointers can support different algorithms than the ones implemented presently both at the differential and the nonlinear equation level. A separate or common time scale for each subcircuit can be used for the discretization of the differential equations together with Newton-Raphson iterations. The Gauss-Seidel or Gauss-Jacobi algorithms can then be employed at the system level in a similar way as used in wave-form relaxation or mixed-mode simulators.

## 6.2. SIMD Processor Role

CLASSIE can run on different computers, however the speed-up described in Chapter 5 is unique for the CRAY-1. On a scalar computer the ordering of semiconductor devices and the definition of vector operations do not produce any speed-up because they are executed in the same sequential manner as before. The only advantage for a typical scalar computer resides in the somewhat reduced data traffic, more locality in the data structure and a reduced search in the linear equation solution.

The first advantage mentioned above is the result of the semiconductor devices ordering by models which saves time in gathering the model parameters.

The second advantage of data storage in separate tables, by element type, prevents the danger of excessive page faulting in the case of a large circuit. This danger exists when the data on the different circuit elements

are stored as linked lists in a single table as is the case in SPICE2. A situation can be imagined where an element is linked to another one of the same type residing in a different memory page and this latter one in turn is linked to a third in yet another memory page.

The third advantage of CLASSIE over SPICE2 on a scalar computer originates in the reduction of the search through the sparse-matrix pointers which is believed to be the cause of the exponential increase of linear equation solution time with the number of equations. From the statistics of Chapter 5 it can be seen that a large matrix, i.e., one with over 100 equations, is replaced in CLASSIE by a few matrices with typically around 20 equations.

The summation of all these effects contribute to a speed-up of roughly a factor of two for the simulation of the Adder4 circuit on a VAX 11/780 computer.

CLASSIE can also be expected to outperform SPICE2 on other vector computers, e.g., the CYBER 205, or array processors such as the FPS 164. Optimal results can be obtained however only if the specifics of each architecture are considered.

A key factor in obtaining top performance out of the CYBER 205 are its specifics as brought out in Chapter 2. These are fast gather/scatter instructions and large vector start-up overhead. The second factor is detrimental for a low number of instances of a certain cell type. The fast gather/scatter suggests that exploitation of latency in this case can be the important source of speed-up. On a CYBER-205 the semiconductor device model evaluation is expected to perform at high Mflops rates due to the large vector lengths achieved. For the linear equation solution it is not obvi-

ous if the use of vector operations can be advantageous considering the reduced size of submatrices and interconnection matrix with less than 100 rows and columns typically.

The FPS-164 with its 12 Mflops is also a good candidate to run CLASSIE efficiently. Running as an attached processor to a general-purpose computer, e.g., the VAX 11/780, the array processor should host only the computational kernel of the simulator. The input, error checking, data structure setup, and output parts can run optimally on the front-end computer. This approach of structuring a circuit simulator in loosely connected parts are pursued independently by various projects involved with the modification of SPICE2, e.g., at Cray Research, [May82], or the development of SPICE3, [Quar82]. The computational kernel of the circuit simulator which runs on the array processor can benefit from the different techniques used in this project. For the FPS-164 the best suited approaches could be a combination of alternatives listed for the CRAY-1 and CYBER 205. The scalar performance of this array processor, judged by SPICE2 statistics, is roughly 4-5 times faster than that of the VAX 11/780 with a floating-point accelerator. As shown in Chapter 3 SPICE2 runs almost 30 times faster on a CRAY-1, in scalar mode, compared to the VAX 11/780.

### **6.3. Salient Results and Future Perspective**

A most important result of the reported work is the speed-up of up to an order of magnitude which can be obtained performing a hierarchical simulation at the transistor level on a vector computer. This speed-up is unique for a circuit simulator and is not available to a timing or logic simulator. The reason for this potential difference in speed is the large number

of floating-point operations in circuit simulation compared to little and almost none in the timing and logic simulation, respectively.

The performance gain has been shown to be a function of the circuit size with increasing speed-ups for larger circuits. A consequence thereof is an approximately linear increase of execution time with circuit complexity. This has been verified on examples as large as one thousand node/equations.

An important goal in this vector-computer-oriented simulator has been to increase the efficiency of data transfers. Statistical measurements taken on CLASSIE prove an important percent increase in the share of the simulation time used by actual floating-point operations. Due to the extremely efficient functional units the SPICE2 simulation time is dominated by memory transfers.

The solution of the linear system of equations is kept below 20% of the total run time for up to 2000 equations (and less devices). This part of the analysis has been believed to use a growing part of the simulation time once the circuit gets larger and is described by more than a few hundred equations. This is achieved by a most efficient sparse equation, machine-code solver using both scalar and vector computation.

All the added performance leads to the prediction that a 5000 time-point transient analysis, with 3 iterations per time point, of a 2000 semiconductor device-node circuit will require 20 minutes of CRAY-1 cpu time based on the run time prediction formula of Chapter 5:

$$T = 5000 \times 3 \times 2000 \times 40\mu s = 20 \text{ min.}$$

To further reduce the run time algorithmic improvements are required.

A wide variety of computer hardware is available on which to run circuit simulation. With present and envisaged 32-bit microprocessors with the performance of a VAX 11/750, circuits of up to 100 device (nodes) will be able to be simulated most efficiently on such personal work stations. For larger size circuits a CLASSIE kernel, resident on a super-computer, can perform a fast and efficient circuit simulation of the description it has received through a high-speed link from the work station.

## APPENDIX 1

### SPICE 2G Small-Size MOSFET Charge Model

#### 1. Small-Size MOSFET Model Considerations in SPICE2

Two models are presently implemented in SPICE2 [Vlad80] which account for small-size effects; the LEVEL=2 model is based on the Frohman-Grove equation with a large number of second-order effects added while the LEVEL=3 is a semi-empirical curve-fitting model [Liu81].

A good example of the difficulties of implementing an analytical model in a circuit simulator is constituted by the charge-oriented model due to Ward and Dutton (WD) [Ward78]. The following derivations are also useful as an example of the practical implementation of a set of device equations into the algorithmic framework described in Section 2.2.

The WD model is based on the actual distribution of charge in the MOS structure and its conservation:

$$Q_C = Q_D + Q_S = -(Q_G + Q_B) \quad (\text{A1.1})$$

where  $Q_G$  is the gate charge,  $Q_B$  the bulk charge, and  $Q_C$  the inversion charge in the conductive channel; this latter charge is split into a drain charge  $Q_D$  and a source charge  $Q_S$ . The current flowing at any terminal of a region can be related to the charge associated with that region, i.e.,

$$i_C = \frac{dQ_C}{dt}$$

$$i_B = \frac{dQ_B}{dt} \quad (A1.2)$$

$$i_S + i_D = \frac{d(Q_S + Q_D)}{dt}$$

The terms of the indefinite admittance matrix of each transistor are found after applying a numerical implicit integration algorithm to the following equation:

$$\int_{t_0}^{t_1} i dt = Q(t_1) - Q(t_0) \quad (A1.3)$$

The charges at any time point are assumed a nonlinear function of only the terminal voltages at that same time point. Due to the complexity involved in finding the partial derivatives, a simple formulation is used for the charges which is adjusted for continuity at the transitions between regions of operation. If the trapezoidal integration formula is used to discretize the differential equations according to Equation 2.2, Equation A1.3 can be rewritten as follows

$$\frac{h}{2}(i_{y,n+1}^{m+1} + i_{y,n}^m) = (Q_{y,n+1}^m - Q_{y,n}^m) + \sum_x \left. \frac{\partial Q_y}{\partial V_x} \right|_{V_{x,n+1}^m} (V_{x,n+1}^{m+1} - V_{x,n+1}^m) \quad (A1.4)$$

where  $x$  stands for GB, DB and SB, and  $y$  for G, B, D and S. The second subscripts indicate the time point and superscripts the iteration. Thus quantities at time point 'n' are known (previous time point), quantities at time point 'n+1' iteration 'm' are also known from the solution of the last iteration at the present time point. Time 'n+1', iteration 'm+1' is the current iteration which, by substituting the above equation into the modified nodal system, has  $V_{x,n+1}^{m+1}$  as a solution.

Capacitances for the small-signal analysis are defined as

$$C_{yx} = \frac{\partial Q_y}{\partial V_x}. \quad (\text{A1.5})$$

It can be noticed that between each pair of nodes there are two capacitances, which in general are different.

Figure A1.1. plots the voltage dependence of the various terminal capacitances associated with the charges  $Q_G$  and  $Q_B$  according to the formulations of the following section. Only six capacitances out of a total of twelve defined by Equation A1.4 are independent due to the charge-conservation principle expressed by Equation A1.1.  $Q_S$  and  $Q_D$  share equal parts of the channel charge  $Q_C$  in the linear region. In saturation  $Q_D$  gets only  $XQC * Q_C$  where  $XQC$  is a model input parameter intended to assign unequal portions of the inversion charge to the drain and source terminals in saturation. The plot shown in Figure A1.1 uses a factor  $XQC=0.5$ . A characteristic of the charge-controlled model is the nonreciprocity of the two capacitances associated with the same pair of terminals.

## 2. Charge Formulations

The charge formulations as first implemented in MSINC [Youn76] and later in HP-SPICE [Dowe79], display two singularities at  $V_{DS} = 0$  and  $V_{GB} = V_T$ . Alternate formulations are used for  $0 \leq V_{DS} < \epsilon$  and  $V_T \leq V_{GB} < V_T + \epsilon$  which avoid an illegal instruction, divide by zero or zero over zero, but still leave discontinuities at the points of joining with the formulation valid beyond these intervals.

In order to keep the charge equation simple the following changes of variables are used throughout this section:

$$V_D = V_{DB} + 2\phi_F$$

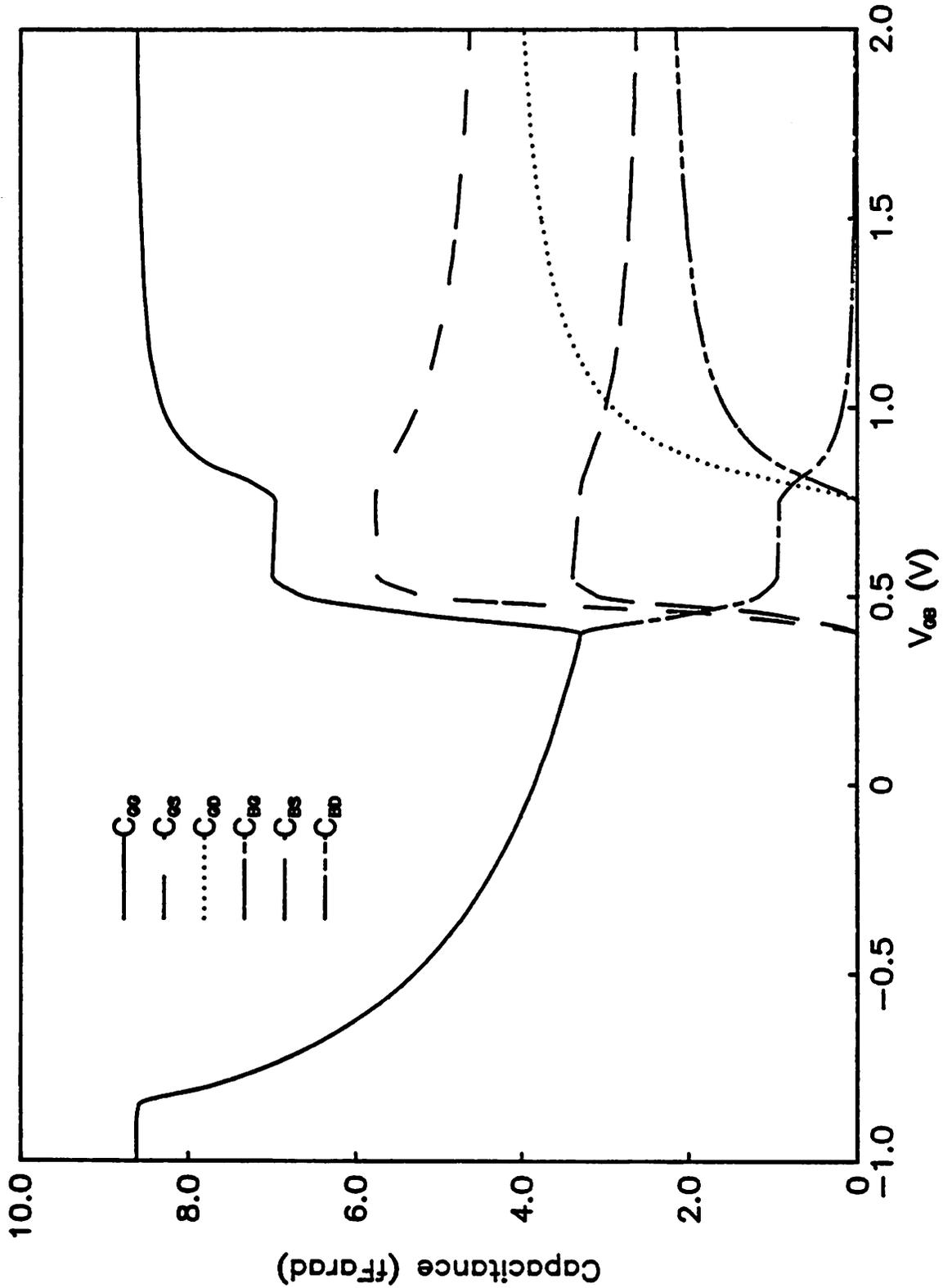


Figure A1.1 Gate- and Bulk-Charge Capacitances of the MOSFET

$$V_G = V_{CB} - V_{FB}$$

$$V_S = V_{SB} + 2\phi_F$$

where  $V_{FB}$  is the flat-band voltage and  $2\phi_F$  the surface potential at the onset of strong inversion. In the charge formulations it is advantageous to choose the bulk voltage as reference voltage.

For the cut-off region,  $V_{CB} \leq V_T$ , and  $V_{CB} \leq V_{FB}$ , below flat-band condition the charges are:

$$Q_G = C_{ox}V_G \quad (A1.6)$$

$$Q_B = -Q_G \quad (A1.7)$$

$$C_{GG} = C_{ox} \quad (A1.8)$$

$$C_{BG} = -C_{GG} \quad (A1.9)$$

$C_{ox}$  is the total thin-oxide capacitance,

$$C_{ox} = WL \frac{\epsilon_{ox}}{t_{ox}}$$

where  $\epsilon_{ox}$  is the oxide permittivity and  $t_{ox}$  the thin-oxide thickness.

Above flat-band,  $V_{CB} > V_{FB}$  the charge and capacitance equations are:

$$Q_G = \gamma C_{ox} \left[ \sqrt{\frac{\gamma^2}{4} + V_G} - \frac{\gamma}{2} \right] \quad (A1.10)$$

$$Q_B = -Q_G \quad (A1.11)$$

$$C_{GG} = \frac{1}{2} \gamma C_{ox} \frac{1}{\sqrt{\frac{\gamma^2}{4} + V_G}} \quad (A1.12)$$

$$C_{BG} = -C_{GG} \quad (A1.13)$$

The channel charge and the rest of the partial derivatives of the gate and bulk charges are zero

$$C_{GD} = C_{BD} = C_{GS} = C_{BS} = 0 \quad (A1.14)$$

as long as the transistor is turned off.

For the turned-on operation of the MOSFET both the original and the corrected equations implemented in SPICE2 are given. In the 'on' region,  $V_{GB} > V_T$ , the following are the original charge equations:

$$Q_G = C_{ox} \left\{ V_G - \frac{1}{I} \left[ \frac{1}{2} V_G (V_E^2 - V_S^2) - \frac{2}{5} \gamma (V_E^{5/2} - V_S^{5/2}) - \frac{1}{3} (V_E^3 - V_S^3) \right] \right\} \quad (A1.15)$$

$$Q_B = -\frac{C_{ox}}{I} \left[ \frac{2}{3} \gamma V_G (V_E^{3/2} - V_S^{3/2}) - \frac{1}{2} \gamma^2 (V_E^2 - V_S^2) - \frac{2}{5} \gamma (V_E^{5/2} - V_S^{5/2}) \right] \quad (A1.16)$$

For the two singularity points mentioned above both the numerator and the denominator of the second term of  $Q_G$  and of the  $Q_B$  expression decrease to zero. The replacement of the equation for the normalized current  $I$  with

$$I = V_G (V_E - V_S) - \frac{1}{2} (V_E^2 - V_S^2) - \frac{2}{3} \gamma (V_E^{3/2} - V_S^{3/2}) \quad (A1.17)$$

where

$$V_E = \begin{cases} V_D & \text{for } V_D < V_{Dsat} \\ V_{Dsat} & \text{for } V_D \geq V_{Dsat} \end{cases}$$

allows the factorization of  $(V_E^{3/2} - V_S^{3/2})$ . The following notations are used for a more concise formulation of the SPICE2 equations:

$$\begin{aligned} T_0 &= V_E + V_S \\ T_1 &= V_E^{3/2} + V_S^{3/2} \\ T_2 &= V_E^{3/2} V_S^{3/2} \\ T_3 &= V_E^2 + V_S^2 \\ T_4 &= V_E V_S \\ T_5 &= T_0 T_1 \\ T_6 &= (T_3 + T_4) + T_2 T_0 \\ T_7 &= (T_3 + T_4) T_1 \end{aligned} \quad (A1.18)$$

Given these definitions, we have the equations:

$$Q_C = C_{ox} \left\{ V_G - \frac{\frac{1}{2} V_G \cdot T_5 - \frac{2}{5} \gamma \cdot T_8 - \frac{1}{3} T_7}{V_G \cdot T_1 - \frac{2}{3} \gamma (T_0 + T_2) - \frac{1}{2} T_5} \right\} \quad (A1.19)$$

$$Q_B = -C_{ox} \gamma \frac{\frac{2}{3} V_G (T_0 + T_2) - \frac{1}{2} \gamma \cdot T_5 - \frac{2}{5} T_8}{V_G \cdot T_1 - \frac{2}{3} \gamma (T_0 + T_2) - \frac{1}{2} T_1 \cdot T_2} \quad (A1.20)$$

The channel charge  $Q_C$  is always computed according to the charge conservation principle expressed by Equation A1.1 and the equivalent conductances result from taking the partial derivatives of Equation A1.19 and A1.20.

## APPENDIX 2

### Benchmark Circuits

The input specification of two benchmarks used in Chapter 3 and Chapter 5 is listed below.

```
UA741 CKT - UA 741 OPERATIONAL AMPLIFIER
.OPT ACCT LVL COD=1 LVLTIM=2
.DC VIN -0.25 0.25 0.005
.TRAN 2.5US 250US
*.OP
VCC 27 0 15
VEE 26 0 -15
VIN 30 0 SIN(0 0.1 10KHZ)
RS1 2 30 1K
RS2 1 0 1K
RF 24 2 100K
R1 10 26 1K
R2 9 26 50K
R3 11 26 1K
R4 12 26 3K
R5 15 17 39K
R6 21 20 40K
R7 14 26 50K
R8 18 26 50
R9 24 25 25
R10 23 24 50
R11 13 26 50K
COMP 22 8 30PF
Q1 3 1 4 QNL
Q2 3 2 5 QNL
Q3 7 6 4 QPL
Q4 8 6 5 QPL
Q5 7 9 10 QNL
Q6 8 9 11 QNL
Q7 27 7 9 QNL
Q8 6 15 12 QNL
Q9 15 15 26 QNL
Q10 3 3 27 QPL
Q11 6 3 27 QPL
Q12 17 17 27 QPL
Q14 22 17 27 QPL
```

```

Q15 22 22 21 QNL
Q16 22 21 20 QNL
Q17 13 13 26 QNL
Q18 27 8 14 QNL
Q19 20 14 18 QNL
Q20 22 23 24 QNL
Q21 13 25 24 QPL
Q22 27 22 23 QNL
Q23 26 20 25 QPL
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
CJC=2PF
+ VA=50)
.MODEL QPL PNP(BF=10 RB=20 TF=1NS TR=20NS CJE=6PF CJC=4PF VA=50)
.PLOT DC V(24)
.PLOT TRAN V(24) V(8)
.END

```

```

ADDER - 4 BIT ALL-NAND-GATE BINARY ADDER
.OPT ACCT LVL COD=1 ITL5=100000 LIMPTS=641 CPTIME=3600
.OPT LVLTIM=1
*.OP
.TRAN 5N 3200N
.SUBCKT NAND 1 2 3 4
* NODES: INPUT(2), OUTPUT, VCC
Q1 9 5 1 QMOD
D1CLAMP 0 1 DMOD
Q2 9 5 2 QMOD
D2CLAMP 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDROP 10 3 DMOD
Q5 3 8 0 QMOD
.ENDS NAND
.SUBCKT ONEBIT 1 2 3 4 5 6
* NODES: INPUT(2), CARRY-IN, OUTPUT, CARRY-OUT, VCC
X1 1 2 7 6 NAND
X2 1 7 8 6 NAND
X3 2 7 9 6 NAND
X4 8 9 10 6 NAND
X5 3 10 11 6 NAND
X6 3 11 12 6 NAND
X7 10 11 13 6 NAND
X8 12 13 4 6 NAND
X9 11 7 5 6 NAND
.ENDS ONEBIT
.SUBCKT TWOBIT 1 2 3 4 5 6 7 8 9
* NODES: INPUT - BIT0(2) / BIT1(2), OUTPUT - BIT0 / BIT1.

```

```

*      CARRY-IN, CARRY-OUT, VCC
X1 1 2 7 5 10 9 ONEBIT
X2 3 4 10 6 8 9 ONEBIT
.ENDS TWOBIT
.SUBCKT FOURBIT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
*  NODES: INPUT - BIT0(2) / BIT1(2) / BIT2(2) / BIT3(2),
*      OUTPUT - BIT0 / BIT1 / BIT2 / BIT3, CARRY-IN, CARRY-OUT, VCC
X1 1 2 3 4 9 10 13 16 15 TWOBIT
X2 5 6 7 8 11 12 16 14 15 TWOBIT
.ENDS FOURBIT
***
***  DEFINE NOMINAL CIRCUIT
***
.MODEL DMOD D
.MODEL QMOD NPN(BF=75 RB=100 CJE=1PF CJC=3PF)
VCC 99 0 DC 5V
VI1A 1 0 PULSE(0 3 0 10NS 10NS 10NS 50NS)
VI1B 2 0 PULSE(0 3 0 10NS 10NS 20NS 100NS)
VI2A 3 0 PULSE(0 3 0 10NS 10NS 40NS 200NS)
VI2B 4 0 PULSE(0 3 0 10NS 10NS 80NS 400NS)
VI3A 5 0 PULSE(0 3 0 10NS 10NS 160NS 800NS)
VI3B 6 0 PULSE(0 3 0 10NS 10NS 320NS 1600NS)
VI4A 7 0 PULSE(0 3 0 10NS 10NS 640NS 3200NS)
VI4B 8 0 PULSE(0 3 0 10NS 10NS 1280NS 6400NS)
X1 1 2 3 4 5 6 7 8 9 10 11 12 0 13 99 FOURBIT
RBT0 9 0 1K
RBT1 10 0 1K
RBT2 11 0 1K
RBT3 12 0 1K
RCOUT 13 0 1K
.PLOT TRAN V(1) V(2) V(9) V(3) V(4) V(10) V(11) (0,5)
.PLOT TRAN V(5) V(6) V(11) V(7) V(8) V(12) V(13) (0,5)
.END

```

## APPENDIX 3

### Subcircuit Definition Table Specification

The subcircuit definition table has a special role in the two-level analysis of CLASSIE. This table contains pointers into a number of tables where related information can be found. In the following list it is differentiated between actual table pointers and offsets within other tables. Certain locations in the table contain two elements; the first description represents the current function in the program and the second, in parentheses, is the future meaning.

LOC	+0	number of subcircuit definition references
	+1	offset in subcircuit definition table of reals
	+2	pointer to node table
	+3	offset to element linked list in IELMNT table
	+4	subcircuit expansion flag (pointer to argument real table)
	+5	unused (pointer to argument integer table)
	+6	offset into resistor integer table
	+7	number of resistors within definition
	+8 to	
	+75	offset and number for all other 35 element types
	+76	offset in user-defined subcircuit-node table
	+77	total number of subcircuit nodes
	+78	number of user-specified subcircuit nodes
	+79	offset into subcircuit initial conditions node table LSICND

- +80 number of subcircuit initial conditions
- +81 total number of subcircuit equations
- +82 number of internal subcircuit equations/variables
- +83 number of subcircuit voltage-defined elements
- +84 number of subcircuit current equations
- +85 total number of state-vector LSXi entries
- +86 offset into row-column swap tables
- +87 offset into subcircuit row-column linked lists
- +88 number of nonzero submatrix entries after reordering
- +89 number of nonzero submatrix entries before reordering
- +90 number of internal nonzero submatrix entries
- +91 pointer to table of submatrix external term locations
- +92 offset into factorization machine-code table MADCMP
- +93 number of machine instructions for submatrix factorization
- +94 offset into back-substitution machine-code table MASOL
- +95 number of machine instructions for submatrix back-substitution
- +96 percentage sparsity of submatrix
- +97 number of floating-point operation for subcircuit solution
- +98 offset into subcircuit matrix entry table LSVN
- +99 offset into subcircuit node-voltage table LSVNIM
- +100 offset into subcircuit state-vector table LSXi
- +101 number of submatrix fill-ins
- +102 stride in LSVN table

## APPENDIX 4

### Adder4 and Adder16 Sample CLASSIE Simulations

Two of the medium and large circuits used as benchmarks for the performance evaluation of CLASSIE in Chapter 5 are the 4-bit and 16-bit bipolar adders. The output of the transient analyses of the two circuits are contained in the attached microfiche. The transient simulation of the Adder4 is complete while that of the Adder16 has been stopped after a specified cp time.

## **APPENDIX 5**

### **Small-Circuit CLASSIE Simulations**

The standard benchmark set of SPICE 2G has been run on CLASSIE and the results have been commented in Section 5.7. The output for the small bipolar and MOS benchmark simulations is contained in the attached microfiche.

## APPENDIX 6

### CLASSIE Program Listing

The source code for a CRAY-1 of program CLASSIE is listed in the attached microfiche.

## REFERENCES

- [Agne80] D.Agnew, "SCAMPER - Circuit Design for the 1980's", *TELESIS*, Vol. 7, No. 3, 1980.
- [Cala79] D.A. Calahan and W.G. Ames, "Vector Processors: Models and Applications," *IEEE Trans.*, Vol. CAS-26, September 1979.
- [Cala80] D.A.Calahan, "Multi-Level Vectorized Sparse Solution of LSI Circuits", *Proc.*, IEEE International Conference on Circuits and Computers, New York, Oct. 1980.
- [Cala81] D.A.Calahan, Private Communication.
- [Char81] A.E.Charlesworth, "An Approach to Scientific Array Processing: The Architectural Design of the AP-120B/FPS-164 Family", *Computer*, Vol. 14, Sept. 1981.
- [Chaw75] B.R.Chawla, H.K.Gummel and P.Kozak, "MOTIS - An MOS Timing Simulator", *IEEE Trans.*, Vol. CAS-22, Dec. 1975.
- [Chua75] L.O.Chua and P.M.Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.
- [Coh86] E.Cohen, "Program Reference for SPICE2", ERL Memo No. ERL-

M592, University of California, Berkeley, June 1978.

- [Coh78] E. Cohen, L. Jensen, A. Vladimirescu and D.O. Pederson, "MICE - A Minicomputer Integrated Circuit Emulator," *Conference Record*, Twelfth Asilomar Conf. on Circ., Syst., & Computers, Pacific Grove, California, Nov. 1978.
- [Coh80] E. Cohen, "Performance Limits of a Dedicated CAD System", *Proc.*, IEEE International Symposium on Circuits and Systems, Houston, Texas, April 1980.
- [Coh81] E. Cohen, "*Performance Limits of Integrated Circuit Simulation on a Dedicated Minicomputer System*", ERL Memo No. UCB/ERL M81/29, University of California, Berkeley, May 1981.
- [CRAY76] CRAY-1 Computer Systems Hardware Description Manual, Publication Number 2240004, CRAY Research, Incorporated, Mendota Heights, Minnesota, 1976.
- [CRAY80] CRAY-1 Fortran (CFT) Reference Manual, Publication Number 2240009, CRAY Research, Incorporated, Mendota Heights, Minnesota, 1980.
- [CRAY82] The CRAY X-MP Series of Computers, Publication MP-0001, CRAY Research, Incorporated, Mendota Heights, Minnesota, 1982.
- [CDC80] CDC CYBER 200 MODEL 203 Computer System Hardware Reference Manual (Preliminary Edition), Publication Number 60258010, Control Data Corporation, St. Paul, Minnesota, May,

1980.

- [DeMa79] H.J.De Man, "Computer-Aided Design for Integrated Circuits: Trying to Bridge the Gap", *IEEE J. Solid-State Circ.*, Vol. SC-14, June 1979.
- [Dowe79] HP SPICE Users' Guide, Hewlett-Packard, Palo Alto, California, 1979.
- [Dowe80] R.Dowell, "Efficient Memory Usage on Mini-Computers Using Fortran", *Digest of Papers*, Comcon, San Francisco, Feb. 1980.
- [Fan77] S. P. Fan, M. Y. Hsueh, A. R. Newton and D. O. Pederson, "MOTIS-C: A New Circuit Simulator for MOS LSI Circuits," *Proc.*, IEEE International Symposium on Circuits and Systems, Phoenix, Arizona, April 1977.
- [Frer76] J.P. Freret, Jr., "Overcoming Wordlength Limitations in Mini-computer Aided Circuit Analysis," Ph.D. Dissertation, Stanford University, Stanford, California, May 1976.
- [Getr76] I.Getreu, *Modeling the Bipolar Transistor*, Tektronix, Inc., Beaverton, Oregon, 1976.
- [Gust67] F. Gustavson, W. Liniger, and R. Willoughby, "Symbolic Generation of an Optimal Crout Algorithm for Sparse Systems of Linear Equations," IBM Report RC 1852, IBM Thomas Watson Research Center, Yorktown Heights, New York, June, 1967.
- [Hach81] G. Hachtel and A. L. Sangiovanni-Vincentelli, "A Survey of Third-Generation Simulation Techniques", *IEEE Proc.*, Vol. 69,

Oct. 1981.

- [Hail82] S.Hailey and K.Hailey, HSPICE is available through Meta-Software, Cupertino, California.
- [Ho75] C.W. Ho, A.E. Ruehli, and P.A. Brennan, "The Modified Nodal Approach to Network Analysis," *IEEE Trans.*, Vol. CAS-22, June 1975.
- [Ho77] C.W.Ho, D.A.Zein, A.E.Ruehli, and P.A.Brennan, "An Algorithm for DC Solutions in an Experimental General Purpose Interactive Circuit Design Program", *IEEE Trans.*, Vol. CAS-24, Aug. 1977.
- [Jenk82] F.Jenkins, ASPEC is available through ISD, Sunnyvale, California.
- [Kasc79] M. J. Kascic, Jr., *Vector Processing on the CYBER 200*, Infotech State of the Art Report "Supercomputers," Infotech International Limited, Maidenhead, UK, 1979.
- [Kell82] K.H.Keller, A.R.Newton, and S.Ellis, "A Symbolic Design System for Integrated Circuits", *Proc.*, 19th Design Automation Conference, Las Vegas, Nevada, June 1982.
- [Kop75] H.Kop, P.Chuang, A.Lachner, and W.J.McCalla, "SLIC - a Comprehensive Nonlinear Circuit Simulation Program", *Conference Record*, Ninth Annual Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, California, Nov. 1975.
- [Kron83] G.Kron, *Diakoptics - Piecewise Solution of Large-Scale Systems*,

MacDonald, London, England, 1963.

- [Lach78] A.F.Lachner and W.J.Mc. Calla, "Node Suppression in the Simulation of IIL", *Proc.*, IEEE International Symposium on Circuits and Systems, N.Y., N.Y., May 1978.
- [Laur76] R.Laur and H.P.Strohband, "Numerical Modeling Technique for Computer-Aided Circuit Design", *Proc.*, IEEE International Symposium on Circuits and Systems, Munich, F.R. Germany, April 1976.
- [Lela82a] E.Lelarasme, A.E.Ruehli, and A.L.Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits", *IEEE Trans.*, Vol. CAD-1, July 1982.
- [Lela82b] E. Lelarasme and A.L. Sangiovanni-Vincentelli, "RELAX: A New Circuit Simulator for Large-Scale MOS Integrated Circuits", *Proc.*, Nineteenth Design Automation Conference, Las Vegas, Nevada, June 1982.
- [Liu81] S. Liu, *A Unified CAD Model for MOSFETs*, Memo UCB/ERL M81/31, Electronics Research Lab., University of California, Berkeley, May 1981.
- [May82] J.C.May, Cray Research SPICE2, private communication.
- [McCa71] W. J. McCalla and D. O. Pederson, "Elements of Computer-Aided Circuit Analysis," *IEEE Trans.*, Vol. CT-18, January 1971.
- [McCa76] W. J. McCalla, *Computer-Aided Circuit Simulation Techniques*, EECS 290H Class Notes, University of California, Berkeley,

Winter 1976.

- [Nage71] L. W. Nagel and R. Rohrer, "Computer Analysis of Nonlinear Circuits, Excluding Radiation (CANCER)," *IEEE J. Solid-State Circ.*, Vol. SC-6, Aug. 1971.
- [Nage75] L.W. Nagel, "*SPICE2 - A Computer Program to Simulate Semiconductor Circuits*", ERL Memo No. ERL-M520, University of California, Berkeley, May 1975.
- [Nage80] L.W. Nagel, "ADVICE for Circuit Simulation", *Proc.*, IEEE International Symposium on Circuits and Systems, Houston, Texas, April 1980.
- [Newt77] A.R. Newton and D.O. Pederson, "Analysis Time, Accuracy, and Memory Requirement tradeoffs in SPICE2," *Conference Record*, Eleventh Asilomar Conf. on Circ., Syst., and Computers, Pacific Grove, California, Nov. 1977.
- [Newt78] A.R. Newton, "*The Simulation of Large-Scale Integrated Circuits*," Memo No. UCB/ERL-M78/52, Electronics Research Laboratory, University of California, Berkeley, July 1978.
- [Newt80] A.R. Newton, "*Timing, Logic and Mixed-Mode Simulation for Large MOS Integrated Circuits*", NATO Advanced Study Institute on Computer Design Aids for VLSI Circuits, Sogesta-Urbino, Italy, July 1980.
- [Quar82] T. Quarles, SPICE3, private communication.
- [Raba79] N.G.B. Rabbat, A.L. Sangiovanni-Vincentelli, and H.Y. Hsieh, "A Multilevel Newton Algorithm with Macromodeling and Latency

for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain", *IEEE Trans.*, Vol. CAS-26, Sept. 1979.

- [Sang77] A.L. Sangiovanni-Vincentelli, L.K. Chen, and L.O. Chua, "An Efficient Heuristic Cluster Algorithm for Tearing Large-Scale Networks", *IEEE Trans.*, Vol. CAS-24, Dec. 1977.
- [Shim82] T. Shima, T. Sugawara, S. Moriyama, and H. Yamada, "Three-Dimensional Table Look-up MOSFET Model for Precise Circuit Simulation", *J. Solid-State Circ.*, Vol. SC-17, June 1982.
- [Tana80] N. Tanabe, H. Nakamura, and K. Kawakita "MOSTAP: An MOS Circuit Simulator for LSI Circuits", *Proc.*, IEEE International Symposium on Circuits and Systems, Houston, Texas, April 1980.
- [Vlad78] A. Vladimirescu, "Accuracy Enhancement in SPICE2 Using Sparse Matrix Pivoting Techniques", EECS 290H Class Project, University of California, Berkeley, Winter 1978.
- [Vlad80] A.Vladimirescu and S.Liu, "The Simulation of MOS Integrated Circuits Using SPICE2", Memo UCB/ERL M80/7, Oct. 1980.
- [Vlad81a] A.Vladimirescu and D.O.Pederson, "A Computer Program for the Analysis of LSI Circuits", *Proc.*, IEEE International Symposium on Circuits and Systems, Chicago, Illinois, April 1981.
- [Vlad81b] A.Vladimirescu, K. Zhang, A.R.Newton, D.O.Pederson, and A.L. Sangiovanni-Vincentelli, "*SPICE Version 2G Users' Guide*", University of California, Berkeley, 10 Aug. 1981.
- [Vlad82] A.Vladimirescu and D.O.Pederson, "Performance Limits of the CLASSIE Circuit Simulation Program", *Proc.*, IEEE International

Symposium on Circuits and Systems, Rome, Italy, May 1982.

- [Ward78] D.E.Ward and R.W.Dutton, "A Charge-Oriented Model for MOS Transistor Capacitances", *IEEE J. Solid-State Circuits*, Vol. SC-13, Oct. 1978.
- [Week73] W.T.Weeks, A.J.Jimenez, G.W.Mahoney, D.Mehta, H.Qassemzadeh, and T.R.Scott, "Algorithms for ASTAP - A Network Analysis Program", *IEEE Trans.*, Vol. CT-20, Nov. 1973.
- [Wolf78] M. J. Wolfe, "Techniques for Improving the Inherent Parallelism in Programs," Report No. UIUCDCS-R-78-929, Department of Computer Science, University of Illinois, Urbana-Champaign, July 1978.
- [Yang80] P.Yang, *An Investigation of Ordering, Tearing, and Latency Algorithms for the Time-Domain Simulation of Large Circuits*", Report R-891, Coordinated Science Lab., University of Illinois, Urbana, Aug. 1980.
- [Youn76] T.K. Young and R.W. Dutton, "Mini-MSINC - A Minicomputer Simulator for MOS Circuits with Modular Built-in Models," *IEEE J. Solid-State Circ.*, Vol. SC-11, Oct. 1976.