ANALYSIS OF PERFORMANCE AND CONVERGENCE ISSUES FOR CIRCUIT SIMULATION

by

Thomas Linwood Quarles

Memorandum No. UCB/ERL M89/42

24 April 1989

ANALYSIS OF PERFORMANCE AND CONVERGENCE ISSUES FOR CIRCUIT SIMULATION

by

Thomas Linwood Quarles

Memorandum No. UCB/ERL M89/42

24 April 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

Acknowledgements

Many people have contributed to the development of SPICE over the years. The work of Laurence Nagel, Ellis Cohen, Sally Liu, Andrei Vladimirescu, and others too numerous to name under the direction of Professor Donald Pederson provided a very solid base from which to more forward.

The research described in this dissertation would not have been possible without the environment for research provided by the CAD research group at Berkeley under Professors Newton, Pederson, Sangiovanni-Vincentelli and Brayton. The availability of equipment, the environment of interaction among a variety of people working on many related projects with the consequent exchange of ideas and the flexibility to follow an idea wherever it may lead have all contributed to the current state of SPICE3.

I would like to express my appreciation to all of the members of the cadgroup, past and present, who have provided valuable discussions and contributions to SPICE3. Particular thanks are due to Jim Kleckner, Res Saleh, Jacob White, Kartikeya Mayaram, and Don Webber for their assistance with the problems of circuit simulation and programming. The contributions of Wayne Christopher and Jeff Hsu in the development of Nutmeg made it possible to concentrate on the simulation aspects of SPICE3 without worrying about the user interface.

This work has been supported by a variety of sources and in a variety of ways. Particular thanks go to Gerry Marino and Raytheon for both financial support and a willingness to try new, relatively unproven software, while providing valuable feedback and ideas along with the expertise needed to make SPICE3 work on IBM PCs. Analog Design Tools, Digital Equipment Corporation, Hewlett Packard, IBM Corporation, Rockwell, the California state MICRO program, and DARPA under contract N00039-C-87-0182 all provided valuable financial and equipment support for the development of SPICE3 and Nutmeg.

The sites which provided valuable feedback during development, including bug reports and fixes, along with many test decks are too numerous to name, but a few stand out for providing considerable amounts of useful feedback at important times. Analog Design Tools provided much valuable feedback as a beta test site during early development of the program, and the IC design group working with Professor Bruce Wooley at Stanford University, especially Bernhard Boser, along with many U.C. Berkeley students, have provided valuable pre-release torture tests for several releases.

Finally, I'd like to thank my parents for their constant support and continuous encouragement to do well at whatever I chose and to continue my education, without which, this dissertation would have been impossible.

Table of Contents

CHAPTER 1

Introduction

SPICE [Nage75a, Cohe76a] is a general-purpose circuit simulation program which accepts a description of a circuit and provides several forms of accurate and detailed simulation, including small signal ac, dc, and time-domain transient solutions. Versions of this program have been in use for almost twenty years, and today there are over 10,000 copies of the program in use world-wide, making it without doubt the most successful single program for electronic circuit design ever developed.

Though integrated circuit technology and design techniques have evolved considerably over the past quarter century, circuit simulation remains a very important component of the design of integrated circuits. Although it may be quite expensive to simulate all sections of a large design at the circuit level, in almost all cases the savings possible if just one incorrect fabrication run is avoided are sufficient to justify extensive and detailed simulation in most cases. SPICE2 has served the electronics community very well, but has been in use for over fifteen years now and is in need of significant cleanup.

While SPICE2 is a solid program and is in wide use, the changes which occurred during fifteen years of incremental maintenance and enhancement of such a large program, without strict change control and detailed documentation, have lead to some inconsistencies and errors. SPICE3 began as a rewrite of SPICE2 using the same basic algorithms which have proven so reliable in SPICE2, but with a new, modular implementation in a different programming language[Quar83a]. Every effort has gone into making this new version as simple as possible to work with, with particular attention being paid to the types of changes that were made most often or were most desired over the lifetime of SPICE2, thus providing a valuable research tool for algorithm design and model development.

Since SPICE2 was first written, many changes have occurred in the field of integrated circuits. Circuits have become much larger and more complex, improved models for the operation of the

semiconductor devices have been developed, and the types of circuits being designed and simulated have changed from predominately bipolar to MOS, CMOS, and even to GaAs MESFET circuits. These technologies have also resulted in new modeling and analysis complexities, such as charge sharing. SPICE2 has been modified extensively, both at U.C. Berkeley and at numerous industrial sites, to handle many of these changes but no complete analysis of these techniques and the additional features needed to make them work well together has been undertaken to date. In this dissertation I examine some of these techniques, their interaction with one another, and their interaction with the basic program design.

The more specific goals of this research were to evaluate approaches to the improvement of convergence in direct-methods transient and dc analysis, while not compromising simulation speed, and to understand the limits in performance of a general-purpose direct-methods circuit simulator running on a general-purpose uniprocessor.

1.1. Design Tradeoffs

There are many tradeoffs that must be made in the development of a program such as SPICE. Some of these tradeoffs are obvious, such as the tradeoff of space for speed by pre-computing and saving many intermediate values, while others are much less obvious, such as the tradeoff of performance for programming simplicity. In each case, there are many possible solutions from which to choose but SPICE3 must pick a fairly small set of them to present to the user, with every additional option to the user producing an extra level of complexity in the program. Because SPICE is a general-purpose circuit simulator and must perform well across a broad range of circuit families, circuit sizes, and analysis options, the choices made must also lead to a robust solution. Picking the correct set of tradeoffs can be as important to making a system such as SPICE3 useful as the correct choice of algorithms. This issue is reviewed throughout this dissertation and the possible choices and the rationale for those choices is presented. In general, the choices have been made conservatively, with an option to loosen them where some significant benefit can be derived from doing so.

1.2. Characterization of the Problem and the Benchmark Set

One of the problems with analyzing a complex engineering tool like SPICE is the difficulty of characterizing the exact problem it is solving. There are many different measures of the performance of a program such as SPICE, not all of them satisfactory. It is impossible to test SPICE against all possible circuits and, due to the complexity of the program and the circuits it is trying to solve, attempting to prove that a given set of circuits is sufficient to demonstrate the general robustness and performance of the program is also very difficult. In addition, differences between hardware platforms in terms of relative performance and precision of floating-point, transcendental functions, memory reference times, and the performance of integer operations can affect the relative performance of SPICE by over 400% as described in Section 2.1.3.1. During the course of this work, a large number of test circuits have been collected from a variety of sources, many submitted specifically because they exhibit particular problems. This set of circuits is detailed in Appendix G. The testing of SPICE3 has been performed using this large base of circuits, ranging from simple test circuits designed to test particular devices or features to large industrial circuits. Clearly, results for all of these circuits can not be presented in every table, since the collection includes over 150 circuits and still growing, but a representative sample is presented, with emphasis on troublesome circuits and large circuits as well as a selection of the better-known SPICE2 benchmarks.

Rather than using a long or perhaps an obscure name for each test circuit, a convention has been developed and a simple name which conveys some relevant circuit details has been assigned to each test circuit. The details of the naming convention for test circuits is described in Appendix G, but the convention is summarized here for future reference in the body of this dissertation. Each circuit has a five component name, consisting of:

- A leading single letter indicating the type of circuit (e.g. N for NMOS, C for CMOS, Q for bipolar).
- The count of the number of active devices (MOSFETs, BJTs, JFETs, GaAsFETs) in the circuit.

- Either D for an all digital, or A for an analog or mixed analog-digital circuit.
- One or more letters indicating the type of simulation performed (e.g. A for ac, T for transient)

For example, C258DT is a CMOS digital circuit containing 258 MOSFETs and the results are stated for transient analysis.

1.3. Special-Purpose Simulators

SPICE is a general-purpose simulator and as such attempts to produce an acceptable solution to any of the wide variety of problems given to it. It is possible, in some cases, to implement a program which can outperform SPICE for a class of problems by taking advantage of special knowledge of the circuit or technology used so as to reduce the simulation time without a significant reduction in simulation accuracy. Such special techniques are not used in SPICE3 in general because of the desire for SPICE3 to handle all circuits and to work well on a broad range of hardware platforms. In fact, as mentioned earlier, a version of SPICE itself optimized for a particular machine or operating system can be expected to show significant performance advantage over the general purpose version developed at U.C. Berkeley.

This is not to say that there isn't a need for special-purpose simulators, or that they are not suitable for the problems they are designed for, but that they do require more care to ensure that the proper tool is being used for the job. For example, for the analysis of a static, CMOS digital logic circuit, where signal coupling between independent nets is negligible, a digital logic simulator may be sufficient to predict circuit behavior and performance. The use of such simulators in conjunction with SPICE3 is strongly encouraged and SPICE3 has been designed with such use in mind. A multi-level simulation system designer should find that the programming interface to SPICE3 is designed to make it easy to use it as a "subcircuit evaluator". It can be used for those subcircuits that require the attention of a general-purpose, accurate, analog simulator within a larger circuit, where other blocks could benefit substantially from a special-purpose simulator such as a logic simulator. This interface for implementing a mixed-mode simulator is outlined in Chapter 4 and details are included in Appendices A through C. The performance differences between SPICE3 and other special-purpose

simulators will become small when the tightly-coupled analog behavior of a circuit (e.g. capacitive or inductive coupling between signals) is important in correct circuit operation as presented in Chapter 5. It is in these cases the SPICE3 performs at its best.

Finally, many of the programming system design considerations which went into SPICE3 apply equally to special-purpose simulators and mixed-level simulators. These simulation systems may find such features as the standardized interface to the front-end package to be useful to them as well since this allows several simulators to present a uniform interface to the user, making it easier for the user to select the proper simulator for the job without worrying about new languages, new circuit descriptions, or new ways of producing output.

1.4. Toolbox Philosophy

SPICE3 is designed using a *toolbox* approach. [Newt81a] Each package of routines is relatively independent of every other package, thus allowing those that maintain and develop the program to select routines which best fit the task at hand from a wide range of available options. For general use, a version can easily be assembled giving the user access to most features of the program. In an environment where space is at a premium, a more restricted version can be assembled by leaving out those routines which may not be needed, including entire analysis packages and device types. When adding additional capabilities to SPICE3, this variety of routines minimizes the code which must be written from scratch by allowing these pre-existing routines to be re-used.

By following this approach from the beginning, SPICE3 includes routines which represent tools which are no longer needed by the current executable because they have been replaced by other routines with a slightly different function or were written entirely for debugging. These routines remain a part of the documented SPICE3 library and can be used whenever a SPICE3 programmer needs such a tool for the development or testing of any part of the program. These routines also provide a guideline for similar routines which may be needed. Of course, these tools and routines are excluded from the program during production use so that both memory utilization and run times are minimized.

1.4.1. Framework for development and testing

SPICE3 has been designed for easy configuration. By changing a relatively small number of routines, different "simulators" can be produced and radical changes can be made to the behavior of the program. In addition, since by far the most common extension to SPICE2 has been the addition of new device models, the interface from the simulator to the device modeling routines has been made as simple as possible to allow new devices and device models to be added to the program in a very short time and without difficulty. For example, the present GaAsFET model was written based on a set of equations presented in [Stat87a] and was added to SPICE3 by S. H. Hwang after only an afternoon of discussion about the program design. Three days of work on his part resulted in a complete working ac, dc, and transient model. It is no longer necessary for a person interested in device modeling to learn about the workings of much of the program, as was the case with SPICE2, but rather to understand the requirements the rest of the system imposes on the relatively small module they must write.

1.5. Identification and Isolation of Modules

By decomposing SPICE3 into modules, all of the routines which handle one aspect of the problem can be grouped together but can be isolated from code which must deal with other parts of the problem. The modules identified in SPICE3 are listed in Figure 1.1.

Command input parsing
Circuit description parsing
User interaction/batch step scheduling
Sparse matrix handling
Simulator coordination/dispatch routines and common numerical algorithms
Analyses (one package for each analysis type)
  operating point
  ac
  dc
  transient
  transfer function
  pole-zero
  sensitivity
Devices (one package for each device type)
  common support routines
  voltage and current sources (independent, voltage, and current controlled)
  resistors
  capacitors
  inductors and mutual inductors
  transmission lines
  uniform distributed R-C lines
  diodes
  bipolar junction transistors
  JFETs
  MESFETs
  voltage and current controlled switches
  level-1 MOSFETs
  level-2 MOSFETs
  level-3 MOSFETs
  BSIM MOSFETs
Graphics
  device independent
  underlying graphics system interface routines (one per graphics system)
    X window system version 10[Gett86a]
    X window system version 11[Sche88a]
    Model Frame Buffer package (MFB)[Bill83a]

Figure 1.1
Major modules in SPICE3

Since each of the modules knows very little of the operations or details of the other modules, the details of the implementation are unimportant as long as the package as a whole maintains a consistent, well documented interface to the rest of the program. One major advantage of this is the ability to divide the program maintenance effort among several people, since the packages can be main-

tained independently. In addition, when improved algorithms are developed, only a single package need be replaced instead of the entire program.

1.6. Integration of Modules into a Whole

Integration of all of the individual modules together into a seamless whole has been accomplished by carefully tailoring each module to the needs and capabilities of the modules with which it will interact. The interfaces between the modules have all been carefully documented and made as regular as possible to minimize confusion and error in use without sacrificing too much efficiency. Thus, the sparse matrix module does not have the flexibility of a more general package such as *Sparse* [Kund88a], but provides all the features that SPICE3 requires internally in a compact and efficient manner. Since all of the packages are self contained, it is also possible to replace them with any other package which provides a superset of the capabilities of the corresponding SPICE3 package as has happened several times, for example, with different versions of SPICE3 and the *Sparse* package over the past two years.

1.7. Overview of this Dissertation

In Chapter 2 techniques for improving the performance of a direct method circuit simulation system are presented. The CPU time characteristics of the SPICE program are examined and techniques to address various parts of the overall run time are described along with results from their application. In Chapter 3, some of the problems of convergence of SPICE are explored. Methods are presented which solve many, although not all, of these convergence problems. An overview of the system architecture of SPICE3 is presented in Chapter 4. The design goals of the system are described, followed by an outline of the structure of the system, focusing on the static and dynamic data structures that form the core of the program. In Chapter 5 the results of the modifications described in Chapters 2 and 3 are presented with an analysis of their usefulness in the final system and a description of their effects on each other and on the overall program. Conclusions and suggestions for future work are presented in Chapter 6.

The overall structure of SPICE3 is shown if Figure 1.2 and is described in detail in Chapter 4. However, the details of each component of the program are included as appendices. First, the interface between the simulator and a front end or higher-level system is presented in Appendix A (represented as "input"data and "output"data in Figure 1.2). This interface allows a single front end to work with several different simulators and a single simulator to work with several different front ends by presenting a standard set of subroutines and data structures on which both the front end and simulator can rely. Appendix B contains a description of the internal data structures used in the SPICE3 core. These structures are presented in detail and assume a familiarity with the system overview presented in Chapter 4. Knowledge of this level of detail of the structures should only be needed by someone modifying the simulator itself.

Component packages used to build SPICE3 are described in detail in Appendix C. All of the packages produced as a part of the simulator except for the device models are described here, along with the SPICE2 style input parser from the front end and shown in Figure 1.2. The remainder of the

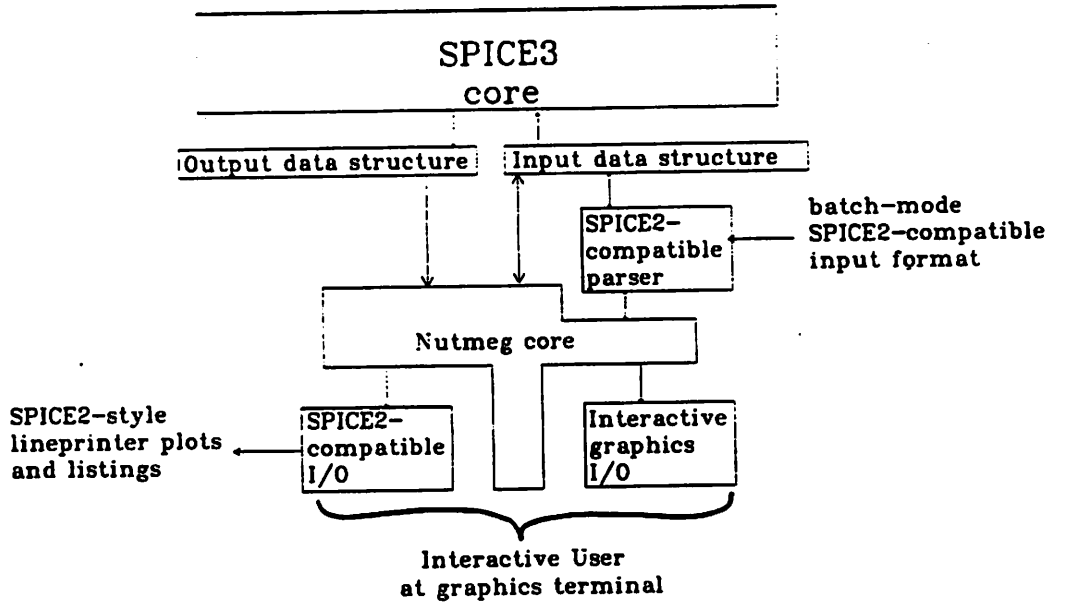

Figure 1.2
The overall structure of SPICE3

front end is described in the Nutmeg Programmers Guide[Chri87a]. Nutmeg is an independent package which is designed to operate as a simulator-independent user interface and post-processor. Nutmeg was developed as a separate project and is not documented here. It serves as an example of a typical user interface to SPICE3.

An overview of the procedure used to add a new device model to SPICE3 is included as Appendix D. This appendix does not contain the details but provides an outline of the steps involved and includes suggestions for converting an existing SPICE2 model to SPICE3 form. Appendix E contains a detailed description of the interface between SPICE3 and the device models. Examples are provided which illustrate typical code for each of the different device model routines. Appendix F provides the users manual for the mostly-SPICE2-compatible input language used to describe circuits and analysis requests.

The test circuits used to verify the operation of SPICE3 are presented in Appendix G along with summary statistics describing each of them. The listings of the circuits themselves are not included in this dissertation, but can be obtained from:

EECS/ERL Industrial Liason Program, Software Office
Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, Ca. 94720.

along with versions of SPICE3 and Nutmeg for a variety of computers, including DEC VAX systems running Ultrix or VMS, HP 9000 systems running HP-UX, SUN workstations, and IBM PC/AT and VM/CMS systems. Please note that the entire benchmark set is not included in the general SPICE3 release and must be requested separately.

CHAPTER 2

Performance

There are many aspects of a circuit simulator that affect its performance and many of these are interdependent (e.g. speed-memory utilization, circuit technology-convergence criteria). This dissertation contains descriptions of the more interesting problems encountered and solutions developed in SPICE3, both those proposed elsewhere and those developed as part of this project. Many additional changes were made in the interest of performance, but many are as simple as sorting tests to place the most likely outcome first, and thus are not described here. The effort has been directed primarily at transient analysis since that represents the majority of the computer time spent running SPICE.

2.1. Where the time is spent

The UNIX system [Ritc74a] provides many tools for analyzing the performance of C [Kern78a] programs[Berk84a, Berk84b, Grah82a]. These have been used extensively to profile the operation of SPICE3 and ensure that the effort spent optimizing the program was directed at the correct parts of the program. The overall solution time can be readily broken down into several significant components and a large number of very small components. For example, a run-time histogram for the 66 node, 116 MOSFET PLA circuit N116DT is shown in Figure 2.1. Unfortunately, the form of this histogram changes significantly with changes in circuit size, analysis requests, and circuit technology. There are, however, a few characteristics of the histogram that are true of most SPICE runs. The time dedicated to all of the system overhead and one-time operations remains relatively small compared to the time for the analysis, matrix package, and per-iteration device code, with the difference between the two groups of times increasing as the analysis interval is increased. The only cases where the time spent in per-iteration and matrix code is not significantly larger than the setup and overhead cost are those with very short analyses which require very little total CPU time. As a result of this, the one time operations can be made relatively simple and easy to understand at the expense of some efficiency

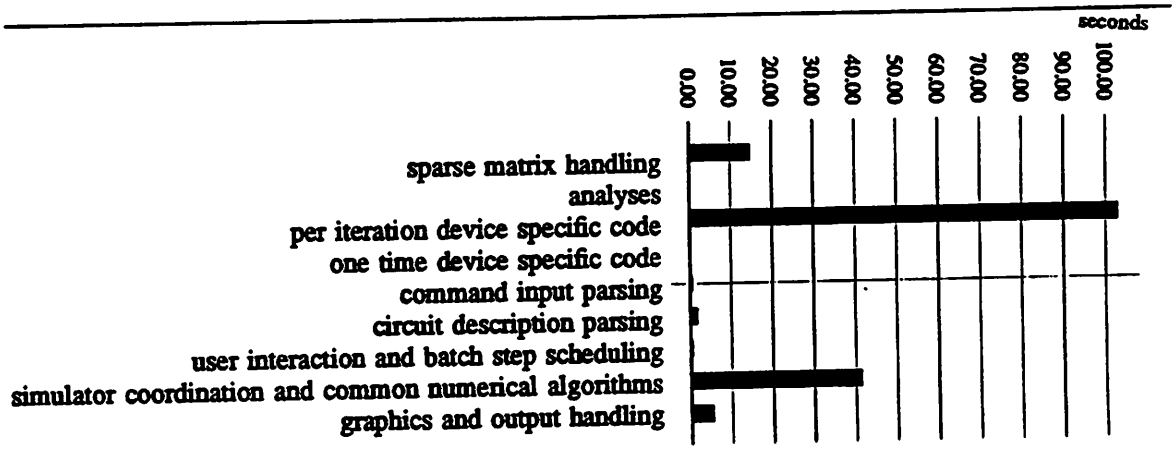

Figure 2.1
CPU time distribution for circuit N116DT

without having a significant impact on overall simulation time.

2.1.1. Per-iteration times

Since the analysis time for large circuits and long simulation intervals is dominated by the total costs of the per-iteration operations, it is appropriate to examine the relative costs of these operations in more detail. In Figures 2.2(a) and 2.2(b), the CPU time per iteration spent evaluating the devices and loading the matrix (*load* time) is plotted along with the total time per iteration. Total time is the sum of load time and matrix solution time (*solve* time) As can be seen from these plots, the matrix solution time is significantly smaller than the device evaluation time, although the difference decreases as the circuit size increases.

Note that Circuit Q340T shown in Figure 2.2 is an exceptional case submitted to Berkeley by an industrial user precisely because of this unusual ratio. Other circuits of comparable size have a lower ratio of load time to total time.

As shown in Figure 2.3 (a)-(d), the growth of the time to evaluate the devices is linearly dependent on the number of devices, since at this level of analysis device interdependence is not considered. The various device types (BJT, MOS, etc) take different times to evaluate and there are slight

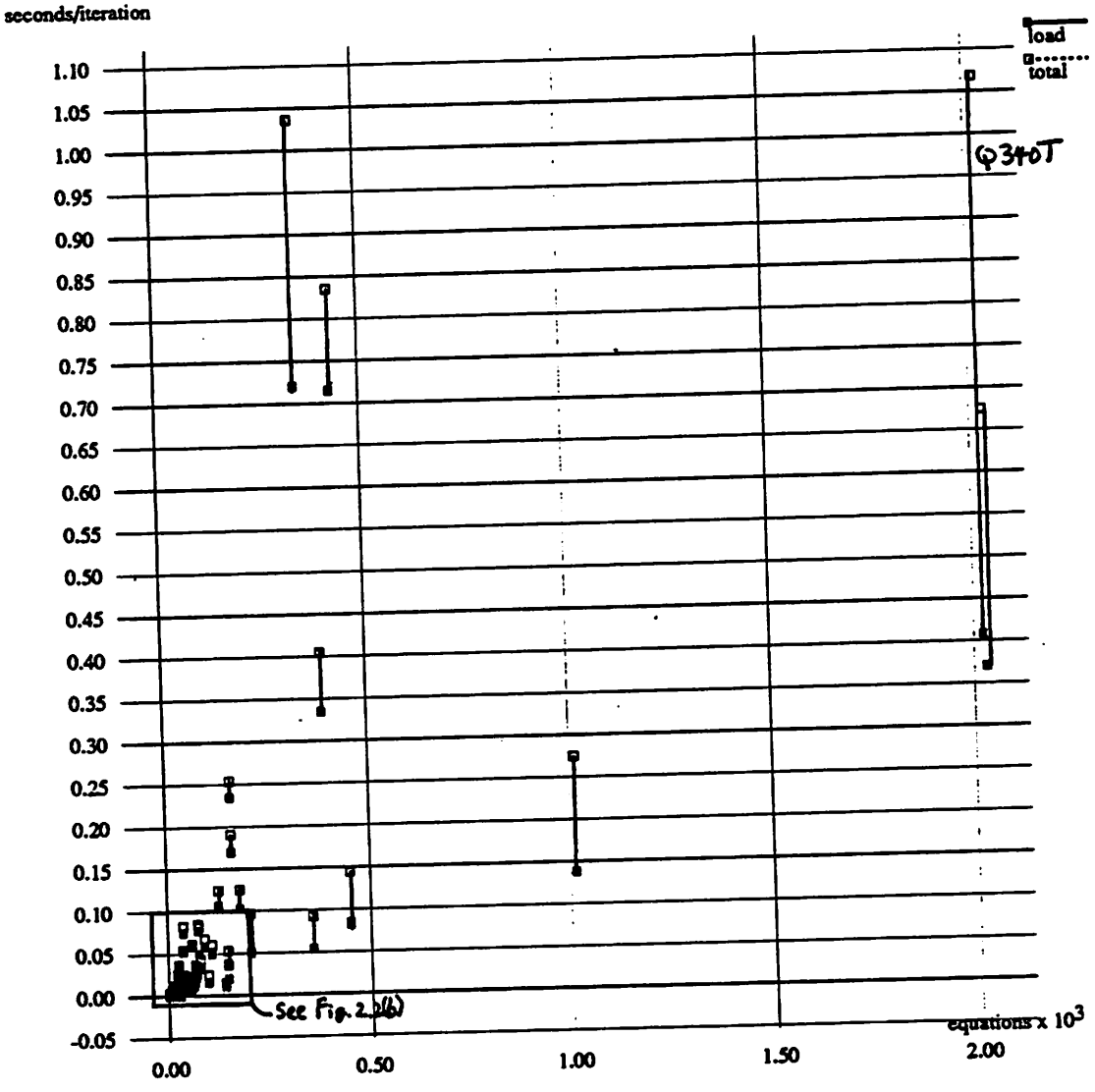

Figure 2.2(a)
Relative Load and Total Times for SPICE3 Versus Number of Equations
(Values for specific circuits are included in Appendix G.)

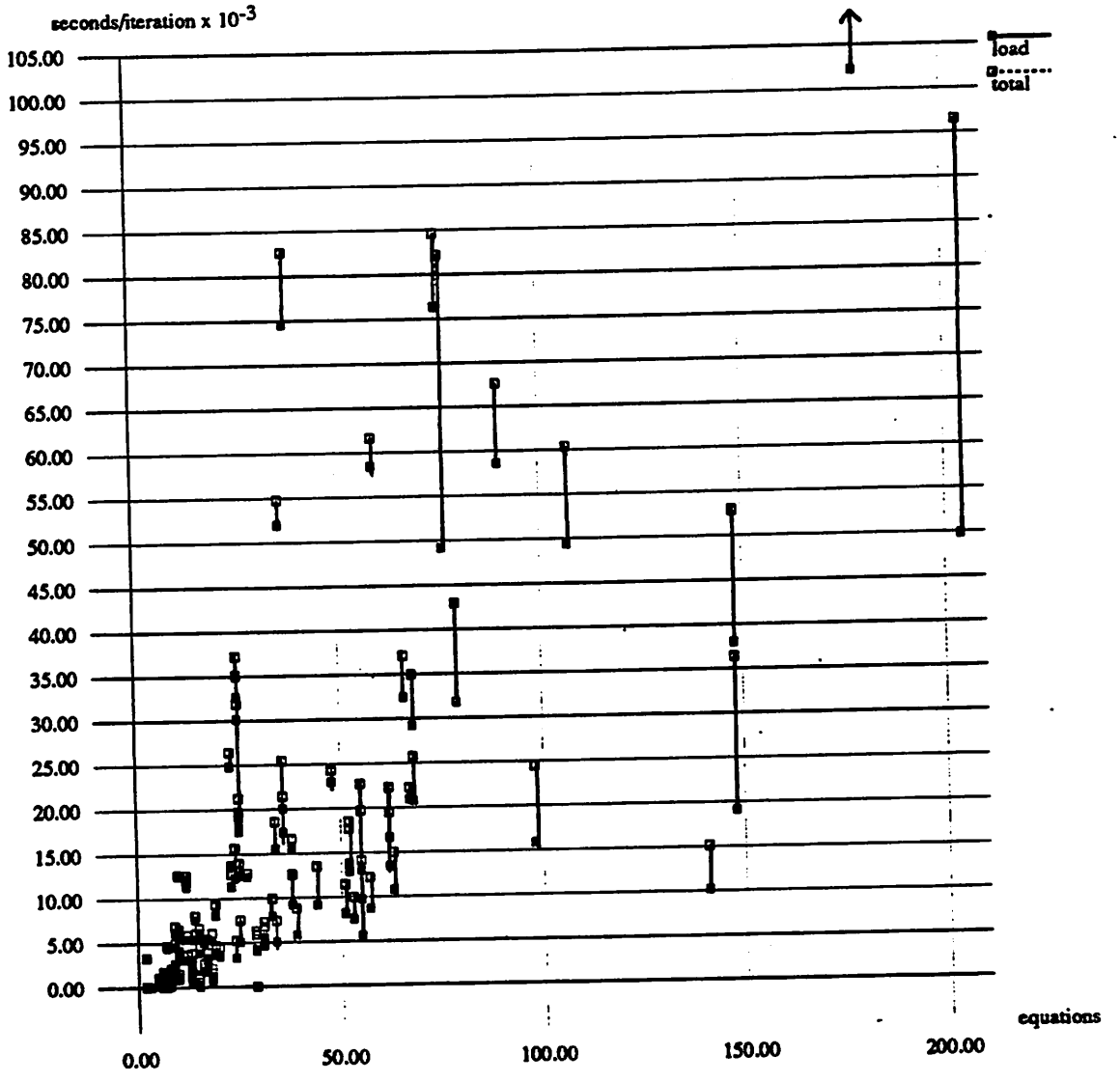

Figure 2.2(b)
Magnification of lower left corner of Figure 2.2(a)

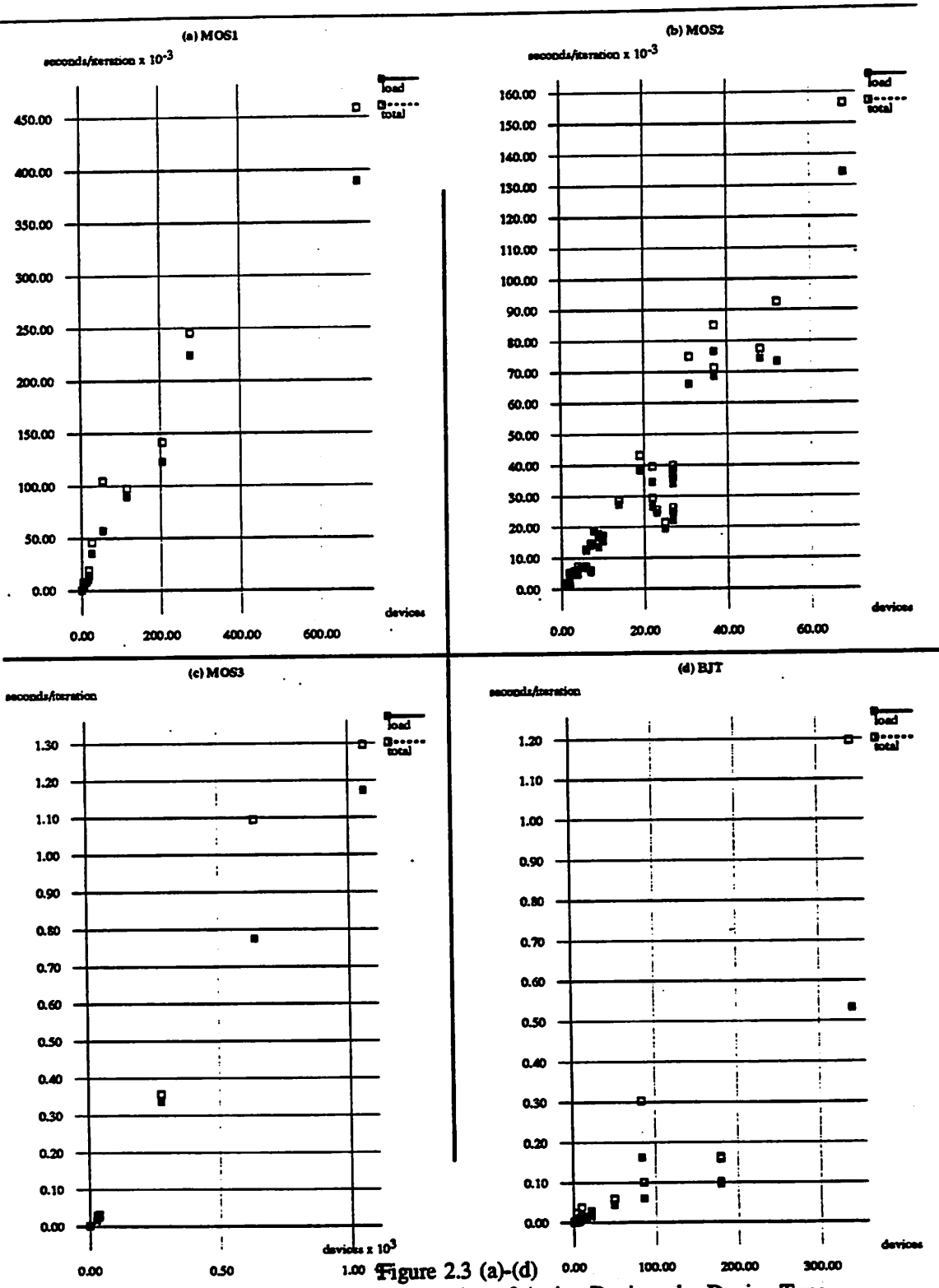

Figure 2.3 (a)-(d) SPICE3 Per-iteration Times Versus Number of Active Devices, by Device Type.

variations due to the different regions of operation of the semiconductor devices, but it takes approximately 0.625 milliseconds/iteration for Level 1 MOSFETs, 1.75 milliseconds/iteration for Level 2 MOSFETS, 1.16 milliseconds/iteration for Level 3 MOSFETs, and 1 milliseconds/iteration for bipolar devices on a VAX 8650 running Ultrix 3.0 and using the standard Ultrix C compiler* and specifying the -O optimization at compile time.

As can be seen by comparing Figure 2.3 with Figure 2.4, the characteristics of the plots have not changed significantly from SPICE2 to SPICE3, although the magnitude of the run time has been reduced by approximately one half from SPICE2 compiled with the standard Ultrix FORTRAN compiler. Compiling SPICE2 with one of the most advanced FORTRAN compilers available today, † the SPICE3 advantage is reduced to approximately 35% on average. In all cases, the load time still represents most of the CPU time.

2.1.2. Matrix solution

At each iteration, the program must solve a set of sparse, linear algebraic equations. While most popular approaches to the problem [Pres86a, Acto70a] require $O(n^3)$ time for a full coefficient matrix of order n, by taking advantage of the sparsity of the matrix obtained in modified nodal analysis[Ho75a] a significant saving can be achieved in time as well as in space. It has been shown [Newt83a] that the solution of a sparse system of circuit equations requires time proportional to n^{β} where n is the order of the matrix and β ranges from 1.1 to 1.5. Figure 2.5 is a plot of matrix solve time versus the number of equations (order of the matrix) for both SPICE2 and SPICE3 running on the benchmark set. Fitting a least-squares line through a log-log plot of this data (ignoring those circuits with solve times less than the timing output tolerance of 0.01) gives solution times of:

$$T_{SPICE2} = 3.01 \times 10^{-5} \times N^{1.40} \times e^{\pm 0.64} \tag{2.1}$$

where e is the base for natural logarithms and N the number of equations, and:

$$T_{SPICE3} = 0.93 \times 10^{-5} \times N^{1.41} \times e^{\pm 0.54} \tag{2.2}$$

* Relative performance on different platforms and for different compilers is presented later.

† The DEC VAX FORTRAN/Ultrix compiler.

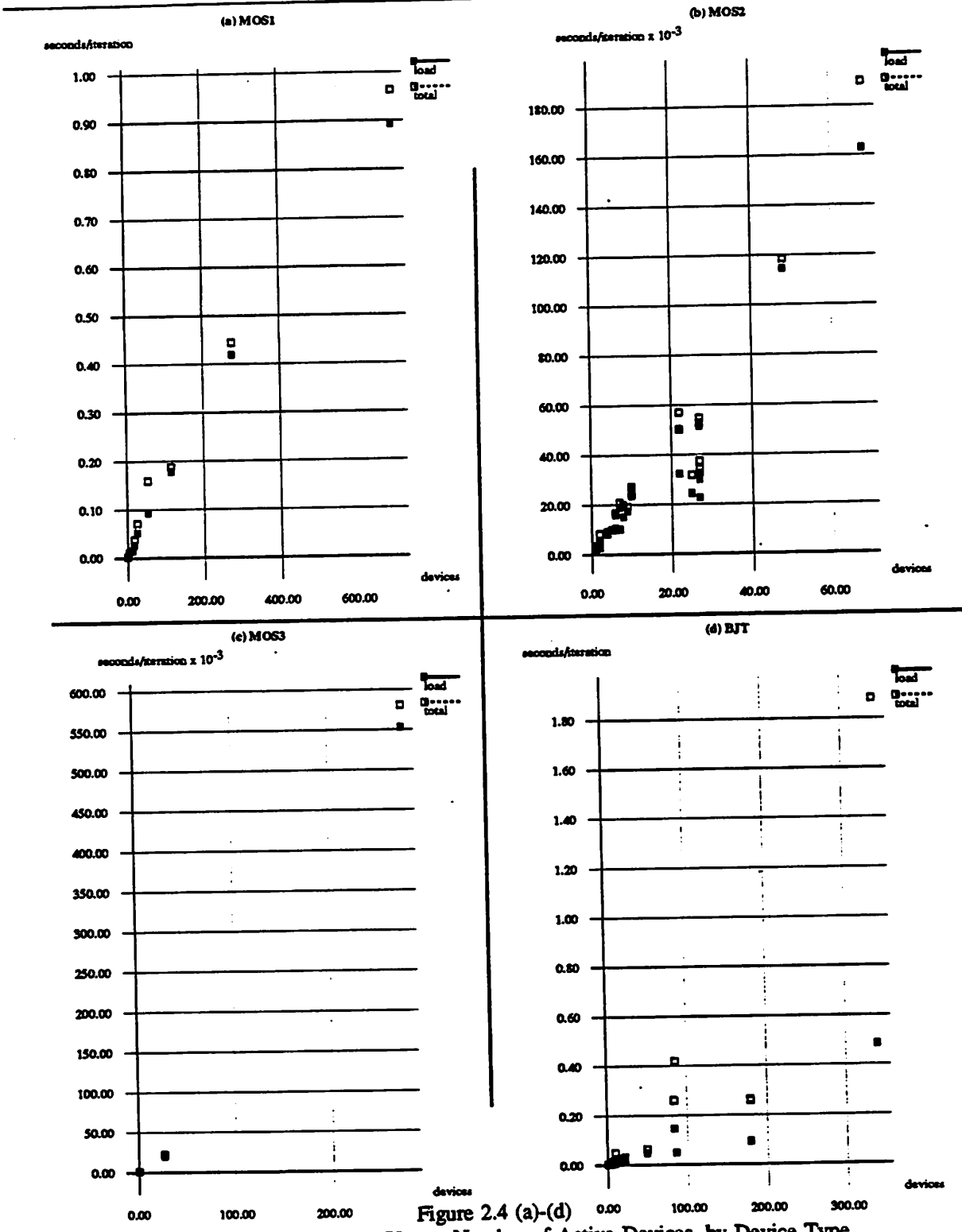

Figure 2.4 (a)-(d) SPICE2 Per-iteration Times Versus Number of Active Devices, by Device Type.

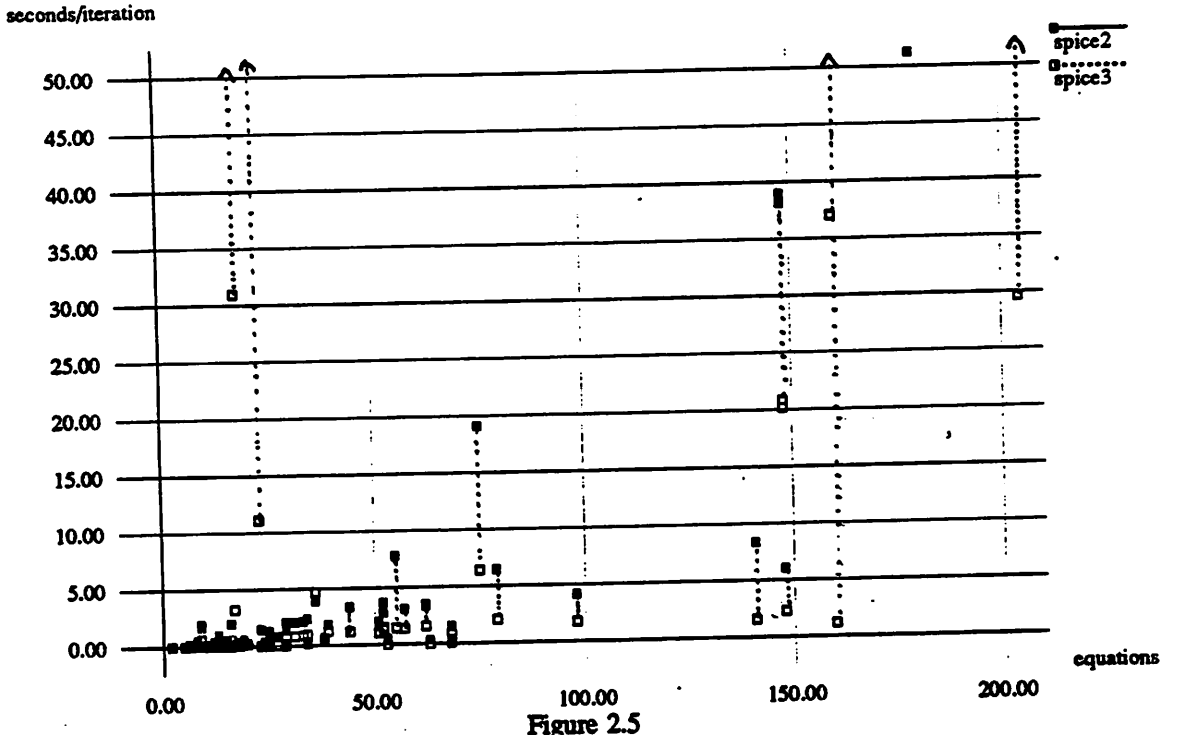

Figure 2.5
Matrix solve time versus number of equations
for SPICE2 and SPICE3.

Thus, while the exponential factor has not changed significantly, given the standard deviation, the leading coefficient has changed by a factor of approximately 3.

While these results do not indicate that the time spent solving the sparse matrices is trivial or can be ignored, its impact on overall simulation time has been reduced as a result of the ability to implement more efficient data structures in C as compared to FORTRAN. Every effort should still be made to make the sparse matrix package as efficient as possible, but sparse matrix solution efficiency is not a prime concern in the remainder of this work, particularly with the availability of general-purpose and efficient packages like SPARSE[Kund88a].

2.1.3. Device evaluation

The other major area where per-iteration time is spent, as shown in Figure 2.1, is the evaluation of device models. While accurate evaluation of the devices is required to maintain simulation

accuracy, there are still tradeoffs that can be made. It may be that there are parts of the device evaluation that use more time than they should due to the way they are implemented, parts of the model that require more time than they may be worth in terms of their contribution to simulation accuracy, or parts that could be "bypassed" based on the specific parameter values for the device in question.

The effort in this work has been focussed on MOSFET devices, since this is the technology being used most widely today. Significant improvement is still possible for bipolar devices as can be seen from the comparisons with an industrial version of SPICE2, presented later. The MOSFET code can be broken into a number of different sections. Most of these sections of code are virtually identical for the Level one, two, and three models, since in SPICE2 they actually shared this code. Looking at these areas carefully is simplest using the Level 1 MOSFET model. A breakdown of the time spent in various parts of this evaluation for an early version of SPICE3 is given in Table 2.6. Clearly, there are parts of this evaluation that take far more time than is to be expected.

2.1.3.1. MOS bulk capacitor evaluation

SPICE includes diodes from source and drain to bulk in the MOSFET models. These diodes have nonlinear capacitances associated with them and SPICE must compute both the capacitance and the charge on the capacitance at each iteration. All equations are presented here in terms of the bulk-to-source quantities, but must be duplicated for bulk-to-drain quantities.

Table 2.6
Breakdown of time spent in MOSFET code for Level 1 Model

Component	Percent of MOSFET time	time(ms/device-iteration)
Gate capacitance	25.2%	0.181
Diodes	33.8%	0.243
Matrix loading	10.2%	0.074
Overhead	12.8%	0.092
Bypass	9.5%	0.068
Evaluation	4.0%	0.029
Limiting	4.5%	0.032

The general form for the equations used for these diode capacitances is:

$$C_{bs}=C_{bs_0}\times(1-\frac{V_{bs}}{P_b})^{-M_j}+C_{bs_{sw_0}}\times(1-\frac{V_{bs}}{P_b})^{-M_{j_{sw}}} \tag{2.3}$$

and:

$$Qbs=P_b\times\left[\frac{C_{bs_0}\times\left[1-\left[1-\frac{V_{bs}}{P_b}\right]\times\left[1-\frac{V_{bs}}{P_b}\right]^{-M_j}\right]}{1-M_j}+\frac{C_{bs_{sw_0}}\times\left[1-\left[1-\frac{V_{bs}}{P_b}\right]\times\left[1-\frac{V_{bs}}{P_b}\right]^{-M_{j_{sw}}}\right]}{1-M_{j_{sw}}}\right] \tag{2.4}$$

These equations encounter severe mathematical difficulties with terms such as:

$$\left[1-\frac{V_{bs}}{P_b}\right]^{-M_j} \tag{2.5}$$

when $V_{bs}\geq P_b$ and M_j is typically $\frac{1}{3}$ to $\frac{1}{2}$. This produces either $\frac{1}{0}$ or the root of a negative number. To avoid this problem, this curve is used up to the point $V_{bs}=F_c\times P_b$, where F_c is a curve fitting parameter between zero and one and P_b is the bulk junction potential. For $V_{bs}>F_c\times P_b$, the capacitance curve is extended as a straight line with the charge curve being correspondingly extended as $\int C\cdot dv$.

In SPICE2 and early versions of SPICE3, the following equations were used:

$$C_{bs}=f3\times\left[\frac{C_{bs_0}}{f2}+\frac{C_{bs_{sw_0}}}{f2}\right]+\frac{V_{bs}}{P_b\times\left[\frac{C_{bs_0}}{f2}\times M_j+\frac{C_{bs_{sw_0}}}{f2}\times M_{j_{sw}}\right]} \tag{2.6}$$

and:

$$Qbs=f1\times\left[\frac{C_{bs_0}}{f2}+\frac{C_{bs_{sw_0}}}{f2}\right]+f2\times\left[V_{bs}-F_c\times P_b\right]\times\left[\frac{C_{bs_0}}{f2}+\frac{C_{bs_{sw_0}}}{f2}\right] \tag{2.7}$$

$$+\left[V_{bs}^2-F_c\times P_b^2\right]\times\left[\frac{C_{bs_0}}{f2}\times M_j+\frac{C_{bs_{sw_0}}}{f2}\times M_{j_{sw}}\right]$$

Where f1,f2, and f3 are coefficients computed during setup. All calculations shown above are performed in this form *every iteration.* By moving to the setup phase all those subexpressions that

have constant value, these can be reduced to:

if Vbs < FCPB

$$V1=\left[1-\frac{V_{bs}}{P_b}\right]^{-M_j} \tag{2.8}$$

$$V2=\left[1-\frac{V_{bs}}{P_b}\right]^{-M_{j_{sw}}} \tag{2.9}$$

$$C_{bs}=C_{bs_0}\times V1+C_{bs_{sw_0}}\times V2 \tag{2.10}$$

$$Qbs=K1\times\left[1-V1\times\left[1-\frac{V_{bs}}{P_b}\right]\right]+K2\times\left[1-V2\times\left[1-\frac{V_{bs}}{P_b}\right]\right] \tag{2.11}$$

if Vbs >FCPB

$$C_{bs}=K3+\frac{V_{bs}}{K4} \tag{2.12}$$

$$Qbs=K5+K3\times\left[V_{bs}-F_c\times P_b\right]+\left[V_{bs}^2-F_c^2P_b^2\right]K6 \tag{2.13}$$

for suitable constants K1,K2,K3,K4,K5, and K6.

This reduces the evaluation time, but the cost of evaluating V1 and V2 is still high. Examining profiling output from early SPICE3 as summarized in Table 2.7, it can be seen that the time spent evaluating V1 and V2 is high relative to the total cost of evaluating the MOSFET due to the use of the exponential and logarithm functions for both bottom and side junctions of both bulk-source and bulk-drain capacitors. Noting that M_j and $M_{j_{sw}}$ may be the same and that the default value for both is 0.5, it is possible to use the square root function in many cases and to eliminate duplicate computations when the grading coefficients are equal. Considering the timing comparison for the square root, exponential, and logarithm shown in Table 2.7, these special cases may be expected to be worthwhile even if used relatively infrequently, depending on the machine used. The implementation of the more complicated math functions varies greatly from machine to machine and a very good exponential and log implementation or a bad square root implementation may reduce the gain significantly, since the exponential plus the log may be only about twice as time consuming as the square root; for most

machines, the gain is much more dramatic.

Table 2.7
Costs of evaluating functions (μs)

	VAX 8650		Microvax II GPX		VAX 8800		DECstation 3100		SUN 4		HP Vectra 386/20 w/387	
sqrt	17.5	1.00	83.5	1.00	17.3	1.00	8.7	1.00	26.7	1.00	41.8	1.00
log	52.4	2.99	325.7	3.90	42.0	2.43	7.8	0.90	22.0	0.82	63.3	1.51
exp	52.6	3.01	371.0	4.44	40.8	2.36	7.2	0.83	22.5	0.84	103.6	2.48

These changes may have a significant impact on overall analysis time as shown in Table 2.8. Additionally, many users when performing early approximate simulations of their circuits or those using external device capacitances will set C_{bs_0} and $C_{bs_{sw_0}}$ to zero, making all of these calculations unnecessary. Adding a further check for this case gives the final column of Table 2.8. Note that this change has the greatest *percentage* effect on the Level 1 model typically used for such approximate simulation.

Table 2.8
Comparison of time spent in capacitor evaluation before and after changes

	With improved formulation	With sqrt	With bypass
Total Simulation time	2647.42	2165.34	2062.83
Total MOS time	1729.60	1285.28	1188.13
Capacitor evaluation time	488.58	116.45	20.54
Time spent in sqrt	0.00	35.47	0.00
Time spent in exp and log	406.00	0.00	0.00
Cap. time as % of MOS time	28	9	2
Cap. time as % of total time	18	5	1

Measurements made on a VAX 8650 running Ultrix 3.0

Finally, note that this analysis was performed using the *original* equations from SPICE2, not the corrected form presented in Chapter 3. This analysis remains essentially unchanged when those corrections are taken into consideration, since they simply modify the constants and not the form of the expressions. The data in Table 2.8, however, was computed using the corrected equations presented in Chapter 3. The column with the results from the original equation formulation has been omitted since, due to the simultaneous correction of the equations as detailed in Chapter 3 and the re-

arrangement of their evaluation as just described, it actually solves a significantly different set of equations and thus it would be inappropriate to compare it directly. The performance for the original equation formulation is essentially the same as for the corrected formulation since the time is dominated by the exponential and log evaluation time which was not affected.

2.2. Predictor-Corrector

Predictor-corrector algorithms are a common means of performing numerical integration which use a predictor, generally a polynomial of low degree, to extrapolate a predicted value from previous points and then successive iterations of a corrector step to refine the solution. There are many benefits to using a predictor-corrector[Acto70a] based algorithm in SPICE3. SPICE2 uses a corrector iteration scheme but lacks a good predictor. A brief analysis of predictors appears in Nagel's dissertation[Nage75a], where the prediction scheme currently implemented in SPICE2 is compared with simply using the solution at the previous timepoint as the starting value for the current timepoint. Unfortunately, the predictor actually used in SPICE2 is not as good as it might be because it is not consistent across the entire circuit as described below.

In SPICE2, the first iteration at each timepoint, is considered a special step, and so is not considered as a possible final solution; at least one more iteration is always taken. In essence, this entire iteration becomes the predictor step but with somewhat unpredictable results. This unpredictability results from the initial conditions of the predictor iteration. SPICE2 assumes that all node voltages in the circuit remain unchanged from the solution at the previous timepoint and most devices are solved under that assumption. Semiconductor devices are assumed to be operating *in isolation from the rest of the circuit*, since their junction voltages *did* change, following a linear extrapolation from their values at previous timepoints. This strange combination of strategies produces a reasonable predictor that proves adequate for SPICE2's Newton-Raphson iteration algorithm to find a solution, despite being based on an often-inconsistent set of assumptions.

By supplying an improved predictor, several benefits are obtained. It has been shown in previous tests [Quar84a] that simply by starting with a better predictor it is possible to reduce significantly the

number of iterations the circuit simulator uses without any impact on accuracy. While more accurate predictors are more expensive to compute, the cost of such a predictor is less than the cost of a single iteration and is therefore worthwhile.

SPICE3 has been fitted with a predictor which performs node voltage prediction and device current prediction using previous solutions and using a predictor of the same order and type as the corresponding corrector. Using this predictor, the performance of the program is as shown in Figure 2.9.

2.2.1. Timestep control

A number of techniques can be used for the control of the simulation timestep. SPICE2 implements two different techniques, called *iteration count* and *truncation error*, to control the timestep

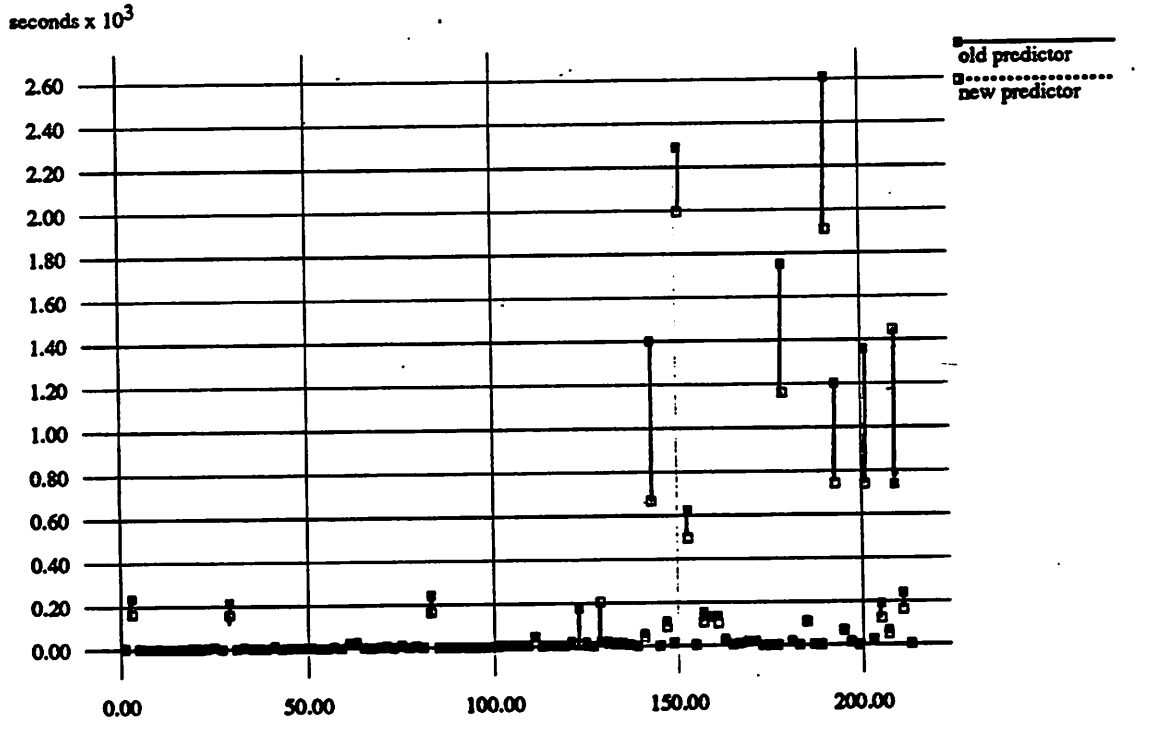

Figure 2.9
Performance gain from the use of a better predictor

between timepoints in the transient analysis. Use of the predictor-corrector algorithm allows a third technique.

While simply using the predictor to reduce the total number of iterations required to solve a circuit is clearly a significant improvement over earlier approaches, further advantage can be taken of the technique. As described by Hachtel and Brayton[Bray72a] , an upper bound on the timestep can be found from the predictor and from the subsequent corrected value after the Newton-Raphson iteration has converged.

2.2.1.1. Iteration count

As described by Nagel[Nage75a], SPICE2 provides an iteration-count system of timestep control. In this system, the number of iterations required to obtain convergence in the damped Newton-Raphson process is used to control the timestep. If convergence is not obtained within a maximum number of iterations, the solution is abandoned, the timestep cut by a factor of eight, and the new smaller step is attempted. If convergence is obtained in fewer than a minimum number of iterations, the timepoint is accepted and the timestep may be doubled before attempting the next step. If convergence is obtained with an iteration count between these limits, the point is accepted and the same timestep used for the next step. This technique relies very heavily on a good choice of the starting timestep by the user and on detecting troublesome areas rapidly. Of course, iteration count does not work at all for linear circuits or almost-linear circuits since only a single Newton-Raphson iteration is required at each timepoint anyway. Unfortunately, the iteration count method does not always detect problem areas and, as observed by Nagel[Nage75a], iteration count does not always produce numerically-acceptable results. For some circuits, particularly for highly-nonlinear *digital* circuits, Newton[Newt77a] shows that iteration count generally provides acceptable results. While still available in SPICE2, this technique is infrequently used and has not been implemented in SPICE3.

2.2.1.2. Truncation error

Truncation error timestep control was also available in SPICE2[Nage75a], and has been provided in SPICE3. In truncation error timestep control, the local truncation error (LTE) associated with the numerical integration of the energy storage elements in the circuit is computed and, based on a limit of acceptable error, used to control the timestep. To compute the LTE, first consider the trapezoidal method and examine the equations for the Taylor series expansion:

$$X_{n+1} = X_n + hX_n' + \frac{h^2}{2}X_n'' + \frac{h^3}{6}X_n''' + \frac{h^4}{24}X_n'''' \cdots \tag{2.14}$$

$$X_{n+1}' = X_n' + hX_n'' + \frac{h^2}{2}X_n''' + \frac{h^3}{6}X_n'''' \cdots \tag{2.15}$$

Solving 2.15 for X_n'' gives:

$$X_n'' = \frac{1}{h}X_{n+1}' - \frac{1}{h}X_n' - \frac{h}{2}X_n''' - \frac{h^2}{6}X_n'''' \cdots \tag{2.16}$$

Substituting 2.16 into 2.14 gives:

$$X_{n+1} = X_n + \frac{h}{2}(X_n' + X_{n+1}') - \frac{h^3}{12}X_n''' - \frac{h^4}{24}X_n'''' \cdots \tag{2.17}$$

The first three terms of the right hand side of this give the trapezoidal rule, with the remaining terms representing the local truncation error of the method. The h^3 term will dominate the other terms, so the LTE can be approximated as:

$$\text{Error} \approx \frac{1}{12}h^3X_n''' \tag{2.18}$$

This is the LTE for this step. Since the allowable error should be divided over all of the steps, the allowable error for the step is:

$$E_{step} = \frac{\text{Error}}{h} \tag{2.19}$$

Thus:

$$E_{step} = \frac{1}{12}h^2X_n''' \tag{2.20}$$

Solving this for h gives:

$$h = \sqrt{\frac{12\,E_{step}}{X_n'''}} \tag{2.21}$$

Using this equation and an approximation to X_n''' obtained from divided differences [Orte70a] provides a limit for the acceptable timestep. If the timestep just taken exceeds this value, its LTE is too large and a smaller step must be taken, with the new value of *h* providing a good estimate for that stepsize.

An iteration count is still used to detect extreme cases where the nonlinear iteration is not converging and a reduced timestep is needed for convergence reasons, but as long as the iteration converges within an upper limit, the truncation error calculation is the only means used to control the timestep of the simulation. This has several disadvantages: the truncation error must be computed separately for each capacitance or inductance and the code to perform the necessary calculations is relatively expensive, requiring approximately 13% of the total analysis time for circuit N698DT as shown in Table 2.10.

2.2.1.3. Predictor-corrector Timestep Control

In the predictor-corrector method[Bray72a], the starting guess at each timepoint is obtained by making a prediction of the node voltage and source current result based on previous timepoints and by using a predictor that is compatible with the following corrector iteration. This predicted value is then used as a first approximate solution and the corrector iteration continues from that point. Once the corrector iteration has converged, the predicted solution and the final corrected solution are compared. The difference between these two solutions gives a good approximation to the local truncation

Table 2.10
Overall simulation time breakdown for circuit N698DT

Component	Percent of total time	time(seconds)
Truncation error	13.3%	341.51
Integration	9.7%	249.15
MOSFET evaluation/loading	57.9%	1480.45
Other evaluation/loading	1.5%	38.64
Matrix operations	5.6%	144.25
Overhead	11.9%	303.13

error E_k of a kth order step as:

$$E_k = \frac{h}{t_{n+1} - t_{n-k}} \times (V_{n+1} - V_{n+1}^P) \tag{2.22}$$

where V_{n+1} is the final corrected value of node voltage at the point in question and V_{n+1}^P is the predicted value of the voltage. h is the timestep taken from time t_n to t_{n+1} and k is the integration order. A version of SPICE3 has been adapted with this algorithm for the Gear integration scheme.

The algorithm has many features to recommend it over the local truncation error computation traditionally performed in SPICE:

- It applies to node voltages and source currents instead of capacitances, thus reducing the number of computations in complex circuits with many MOSFETs, each containing several internal capacitances.
- It employs a consistent predictor, already shown to reduce analysis time.
- It simplifies the device models since they no longer need to include truncation error calculations.
- It is compatible with a node based capacitance and charge computation scheme.

Unfortunately, the predictor-corrector scheme is not quite as helpful as it seems. While suitable predictors and correctors are available for the Gear integration method, no suitable predictor is available for the trapezoidal algorithm most commonly used in SPICE. Using Gear integration, performance is as observed in Table 2.11.

In this table the result of running several circuits from the benchmark set through SPICE3 with both the standard local truncation error timestep control scheme and the predictor-corrector based timestep control scheme are presented. These circuits were run for a variety of relative tolerances to show the effect of the tolerance on the two timestep control schemes. All of the missing entries in the table, except for the one with the standard version with a RELTOL of 0.0001 which was not performed, correspond to runs which failed due to timestep too small errors.

Table 2.11
Performance gain from use of Gear predictor-corrector timestep control

Version	RELTOL	Q11ATAD Iters	Q11ATAD Time-points	Q11ATAD CPU time	Q50AT Iters	Q50AT Time-points	Q50AT CPU time
Standard	0.0001	1576	462	24.56	-	-	-
Standard	0.001	587	174	9.74	3154	1001	238.1
Pred-Corr.	0.001	1515	544	25.75	-	-	-
Pred-Corr.	0.002	1294	440	20.91	-	-	-
Pred-Corr.	0.004	1038	349	16.91	-	-	-
Pred-Corr.	0.01	804	265	13.47	-	-	-
Pred-Corr.	0.02	596	184	9.93	2432	680	176.26
Pred-Corr.	0.1	218	61	3.60	1493	280	100.38

Version	RELTOL	Q5DTD Iters	Q5DTD Time-points	Q5DTD CPU time	Q22ATAD Iters	Q22ATAD Time-points	Q22ATAD CPU time
Standard	0.001	469	121	4.86	183	59	6.86
Pred-Corr.	0.001	-	-	-	436	166	15.35
Pred-Corr.	0.002	-	-	-	595	193	20.30
Pred-Corr.	0.004	-	-	-	4423	1425	146.23
Pred-Corr.	0.01	609	192	7.69	351	116	11.91
Pred-Corr.	0.02	487	135	5.30	239	73	8.13
Pred-Corr.	0.1	383	98	4.15	205	61	7.12

Version	RELTOL	N27AT Iters	N27AT Time-points	N27AT CPU time
Standard	0.001	306	139	13.18
Pred-Corr.	0.001	880	424	33.37
Pred-Corr.	0.002	518	246	19.89
Pred-Corr.	0.004	431	204	15.97
Pred-Corr.	0.01	320	143	12.20
Pred-Corr.	0.02	292	131	10.92
Pred-Corr.	0.1	266	118	10.14

Note that while the predictor-corrector timestep controlled version takes significantly longer to solve for a given set of tolerances, it actually generates a much more accurate result, with the result for a relative tolerance, RELTOL, of 0.002 producing a better waveform than the standard timestep control algorithm does for a RELTOL of 0.0001 for circuit Q11ATAD. With this consideration, the values obtained for the standard method in Figure 2.11 should be compared with those obtained for the predictor-corrector method with a larger tolerance, resulting in better overall performance for the predictor-corrector method in many cases. Unfortunately, convergence problems with this method,

caused by the higher accuracy for looser tolerances, must still be solved before it can be provided for general use.

2.2.2. Limiting

Devices containing exponential curves require some degree of *limiting* [Nage75a] to prevent numerical overflow when proceeding along the exponential. During the sequence of Newton-Raphson iterations required to find a solution, it is possible for a relatively small change in a controlling variable to cause a sufficiently large shift in another exponentially-controlled variable to make a large change in the solution in the next iteration. This change may include a large change in the voltage across a junction. The resulting exponential current increase could easily cause numerical overflow. The solution of the circuit, if it can be computed, will *not* include such a point, so the value can be artificially limited to a value which prevents the overflow as long as at least one additional iteration is carried out to ensure that the small inconsistency in the circuit caused by the limiting does not remain in the solution, thus generating a "damped" Newton-Raphson iteration. SPICE [Nage75a] has traditionally achieved this by limiting the *change* in each junction voltage individually. However, when a junction voltage change is limited, the voltages seen by that junction are inconsistent with those seen by the rest of the circuit, so a limiting operation for each junction in the circuit is required. It has been suggested[Whit85a, Sale87a] that the voltage change limiting should be performed on a per-node basis. This can reduce the time spent in the limiting code, but limiting the node voltage change enough to prevent problems from exponentials reduces the rate at which all of the circuit nodes can change between iterations to an unacceptably low level. This greatly increasing the total number of iterations needed, particularly for dc solutions when nodes controlled by sources may have to rise to 5V in steps of size nV_t for some small value of n. A similar problem occurs when input voltage sources ramp up or down rapidly. Further, the node-based schemes can not take advantage of knowledge of the regions of operation of the devices attached to the node in question, something which the junction-based scheme can handle since the junction-based approach basically limits each junction independently while the node-based scheme has no knowledge of the junction voltages at each node, only the

absolute value of the node voltage itself. Experimentation with the node-based algorithms of RELAX and SPLICE showed a greatly increased iteration count and a corresponding increase in CPU time over the junction limiting technique, thus the node-based schemes were dropped from further consideration.

The predictor-corrector technique appears to offer a reasonable approach to limiting since it can bound the range of solutions that will be acceptably-close to the predictor so as to satisfy the truncation error limit requirement. While this is a node-based limit, it is based on actual circuit activity instead of being a simple fixed limit. Unfortunately, this technique does not help in the dc solution where most overflow problems occur, thus the original technique used in SPICE2 was restored.

2.3. Breakpoints

SPICE has traditionally used a breakpoint table to flag timepoints during the transient analysis which require special attention due to a rapid change taking place[Cohe76a]. When the program encounters a point in a waveform such as the point labeled "a" in Figure 2.12, a derivative discontinuity exists. While the timestep control algorithm used in SPICE2 and SPICE3 will work at this point, experimental evidence has shown that the program will eventually hit a timepoint almost exactly coinciding with points such as this one through a succession of rejected timepoints. Identify-

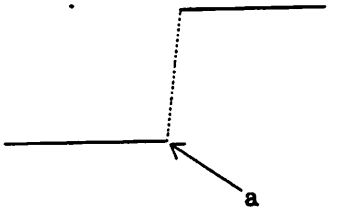

Figure 2.12
An example breakpoint

ing these points in advance and not trying to avoid them can save the time used to generate these potential rejected points. To implement this, SPICE2 maintains a sorted table of breakpoints, generated during the setup phase, and whenever a timestep would carry it *past* such a point, the timestep is cut to fall exactly on the point. This works well for all statically determined breakpoints, such as those produced by piecewise-linear voltage sources, but does nothing for points like this that are not predicted during the setup phase. The only possible source for such sharp transitions as this other than the input sources is the ideal transmission line model. A transmission line can take a breakpoint from a source and delay it, releasing it into the circuit with just as sharp an edge at a later time. To handle these points, SPICE2 takes the precaution of assuming that every breakpoint in the circuit will propagate through every transmission line, generating a breakpoint when it emerges. This breakpoint would then traverse every transmission line again, including the one that just created it, until all possible combinations that might occur before the simulation time limit is reached have been generated. The computation of all such points is performed during setup to ensure that all possible breakpoints are in the table, thus a single sorted table is generated before simulation begins.

The breakpoint table as implemented in SPICE2 can be quite expensive in the case of transmission lines and potentially forces the program to solve the circuit at many new timepoints. In an attempt to minimize these costs, several approaches were considered for reducing the breakpoint cost in SPICE3.

2.3.1. V/I smoothing

Most breakpoints are readily identified as the points at which the slopes of voltage or current source waveforms change abruptly. Since the change in timestep to hit the breakpoint exactly, the complexity of cutting the timestep and integration order, and the additional complexity involved in maintaining the breakpoint table are not trivial, an attempt was made to remove the breakpoints by smoothing out the curves.

In experiments conducted by Shankar Narayanaswamy[Nara87a], the breakpoints were replaced by a quadratic curve fitted to match the piecewise linear segments in position and slope at a point 20%

of the way down the shorter of the two segments meeting at the corner in question and at a point an equal distance down the other segment. This produces the curved segment shown in Figure 2.13. This approach appears to solve the problem, but the change in the derivative of the signal is still too rapid, and one or more points are still required in a relatively narrow range about the breakpoint. Further, the additional cost of the curve fitting increases the total analysis time for most circuits, thus leading to a return to the simple breakpoint table scheme used in SPICE2, although modified slightly as described in the next two sections.

2.3.2. Transmission line problems

For most circuits, the SPICE2 breakpoint scheme is adequate, but if transmission lines are involved, serious problems may result. This approach, while quite conservative numerically, produces serious problems in practice, since typical circuits have several transmission lines and several independent sources of breakpoints, thus producing a very large number of total breakpoints through their assumed interaction in all possible combinations. An additional problem with this approach is it requires that the end of simulated time be known in advance.

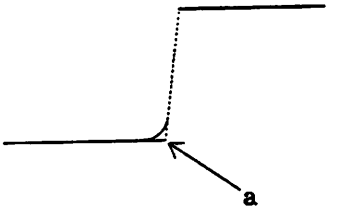

Figure 2.13
Quadratic curve fitted around a breakpoint

Table 2.14
Comparison of static and dynamic breakpoint table sizes and resulting analysis times

Circuit Name	SPICE2 breakpoint table size	SPICE3 breakpoint table size	SPICE2 total CPU time	SPICE3 total CPU time
T2AT	288	24	1.57	0.47
O66AT	1408	24	30.57	10.32
T1ATAD	80	24	1.50	0.78
T2A1T	640	56	2.70	1.47
T1A1T	176	32	0.92	0.54
T1A2T	176	24	0.90	0.32
T3AT	144	16	1.93	1.51
T3A2T	144	16	1.83	1.52
Q4A4TA/Q4A5TA†	286792	16	4096.12‡	49.27
T2A2T	96	16	0.77	0.61

† The same circuit with two different input formats.
‡ aborted due to iteration count. Completed 2.04 ns of a 20 ns simulation.
81.15 seconds of transient analysis, 4012.57 seconds setting up
the circuit for simulation, most spent setting up the breakpoint table.

2.3.3. Dynamic table

In SPICE3, the breakpoint table has been made dynamic; it is constructed as the simulation proceeds and includes only data needed in the immediate future. After each breakpoint is used, it is discarded and, as each source passes a breakpoint, its next breakpoint is put on the list. This handles all the ordinary sources, but the sources added due to transmission lines passing source breakpoints must still be handled. The major problem with the scheme used in SPICE2 is the excessive number of false breakpoints added due to the incorrect assumption that all breakpoints would propagate through all transmission lines. SPICE3 makes the assumption that relatively few of the breakpoints actually propagate through many of the transmission lines. At each tentatively-accepted time point, each transmission line is checked to see if, by a divided-difference calculation, the approximate second derivative at the previous time point is large and, if so, assumes it is a breakpoint to be propagated through that line. Using both relative and absolute criteria, the sensitivity of this can be adjusted from generating a breakpoint for every timepoint to never generating a breakpoint. The cost of this scheme is slightly greater than that for the SPICE2 scheme, since it requires a computation at each timepoint for each transmission line and a computation at each breakpoint for each breakpoint-generating source, as well as requiring an insertion sort on the breakpoint table. The significantly reduced size of

the breakpoint table leads to shorter sorting times, and the problem of false breakpoints from transmission lines is reduced. As seen in Table 2.14, the time saving can be quite dramatic.

2.4. Algorithm Reorganization

The organization of the algorithms in SPICE2 is not necessarily optimal, even where the choice of algorithm has been made very carefully. In some cases, the algorithms appear to have been organized in such a manner as to minimize the number of subroutines rather than to minimize computation.

2.4.1. Convergence check

As an example of this problem, consider the outer loop of SPICE's Newton-Raphson iteration. In SPICE2, the loop appears as in Figure 2.15. In this loop, the *Load and check device convergence* block represents a single call to the LOAD subroutine. In this subroutine, each device in the entire

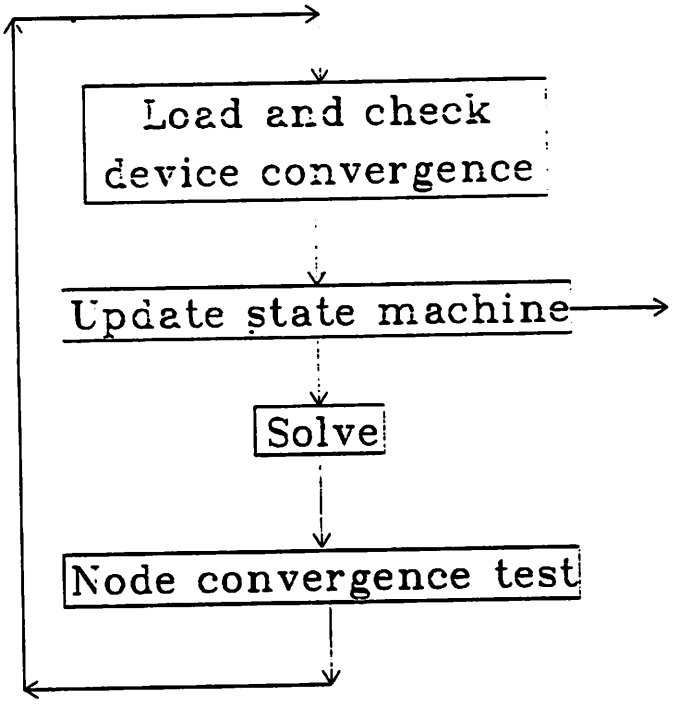

Figure 2.15
SPICE2 Newton-Raphson loop

circuit must be evaluated and as a side effect of the evaluation its convergence criteria is checked. When SPICE2 reaches the *update state machine* step, if convergence was obtained in both the device convergence and node convergence tests, then the sparse matrix loading that was performed during the evaluation is ignored and the loop exits. If the convergence test failed at any point during the node or device convergence test, the rest of the convergence test results are unnecessary and ignored. Modifying this algorithm to reduce these redundant computations results in the flow shown in Figure 2.16, which is the algorithm used in SPICE3. In this algorithm, every time the devices are evaluated the circuit is immediately solved using the resulting matrix. The convergence tests are performed as separate steps and as a result, as soon as either the node or device convergence test fails, the remainder of the tests can be skipped since no other code will be skipped as a result. Finally, since the last evaluation of the devices for which no subsequent solution was performed is eliminated, one less evaluation of the devices per timepoint should be required. While this could conceivably reduce

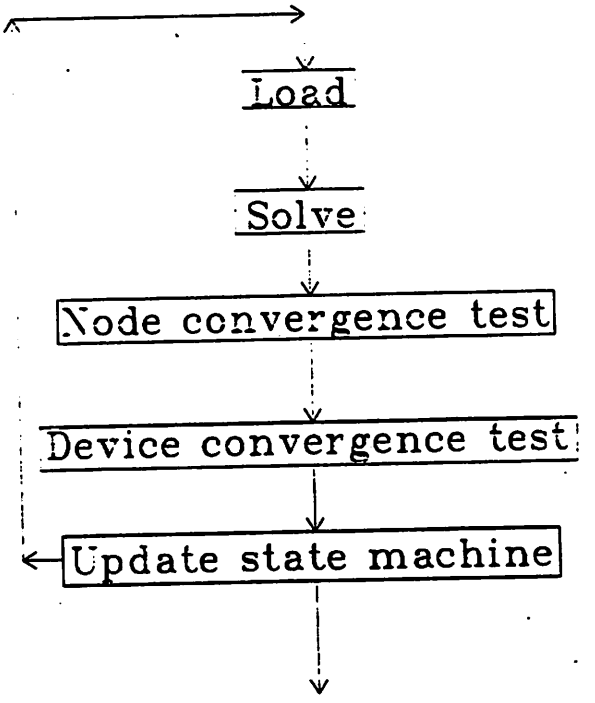

Figure 2.16
SPICE3 Newton-Raphson loop

the number of device evaluations by approximately 25%, since approximately 3 to 4 evaluations are typically required at each timepoint, the actual saving is much less due to a variety of factors. Circuits frequently pass the node convergence test before the device convergence test, thus requiring the of device evaluations and convergence tests be, and in some cases a significant number of iterations are the result of points at which convergence is never obtained. Thus, the savings is significantly less than might be expected, but it is still a significant saving for some circuits and leaves the matrix decomposed for additional operations which may be desired, such as sensitivity analysis.

2.5. Bypass

In a system such as SPICE which requires evaluation of complex models, any algorithm which offers to reduce the number of such evaluations is of great interest. Bypass

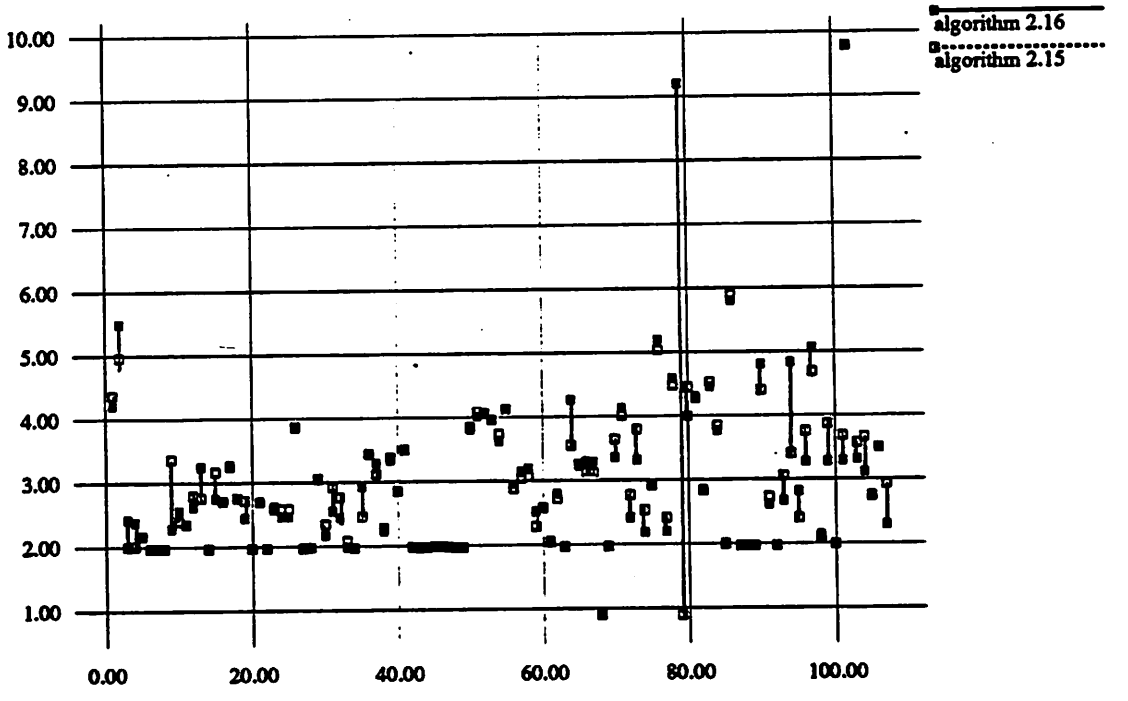

Figure 2.17
Comparison of iterations/timepoint
for algorithms of Figures 2.15 and 2.16

schemes[Nage75a, Webb86a, Sale86a, Newt77a] offer such a reduction by allowing a previously calculated result to substitute for one which otherwise would have to be computed. This technique can be applied at several different levels in circuit simulation as described in the following sections.

2.5.1. Device bypass

A technique which has been employed by SPICE2 since it was first implemented[Nage75a] is *device bypass*. In this technique, a device whose inputs have not changed significantly from the previous iteration is not re-evaluated, but instead the computation is bypassed and the saved Jacobian and right hand side entries from the previous evaluation of the device are re-used. As described by Newton[Newt77a], this technique works quite well for digital circuits since many parts of such a circuit are quiescent much of the time, thus allowing a significant savings in the device evaluation time. The additional cost of evaluating the bypass condition my actually increase simulation time in some cases, primarily very small circuits where every device may be actively changing at every timepoint. However, as shown in Figure 2.18 this bypass technique can significantly reduce the computation time for many circuits. In this plot, the average CPU time per-iteration, both with and without bypass, is shown for each circuit in the benchmark set plotted.

2.5.2. Jacobian bypass

By applying bypass at a different level, it may be possible in some cases to avoid all device evaluations and to re-use the factored Jacobian from the previous iteration[Anto68a]. The exact Jacobian is not needed at every step, simply a good approximation to it. This approach has proven effective in other programs[Whit85a], and it has been suggested that it would work well in SPICE3. Unfortunately, there is a problem with implementing this directly in SPICE3. SPICE has traditionally taken the approach of computing the solution vector directly while some other simulators, such as RELAX, have taken the approach of computing the *change* in the solution vector at each iteration. Given this decision, an analysis of the structure of the equations reveals that the SPICE equation formulation makes the bypass of the entire Jacobian evaluation impractical.

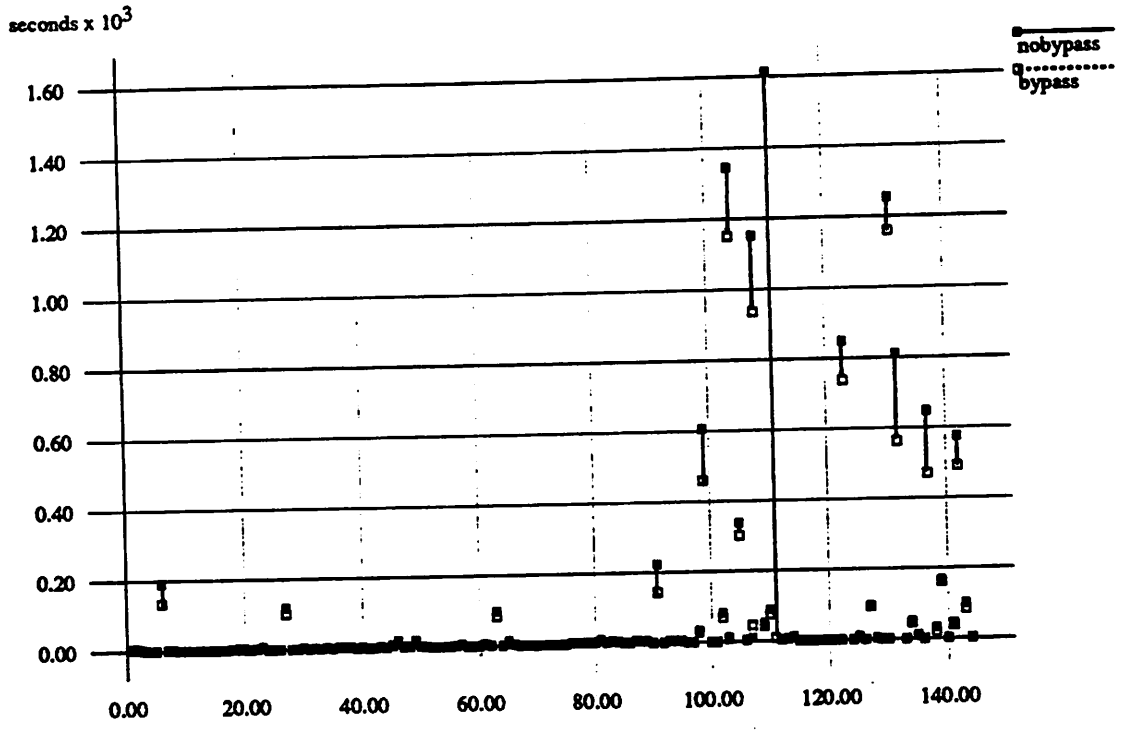

Figure 2.18
Device load time with and without bypass.

The solution being solved in general for the modified nodal analysis is:

$$f(V)=G(V)\cdot V-I(V)=0 \tag{2.23}$$

where V is the desired solution (voltage) vector, I(V) is the vector of independent source currents, and G(V) is the nodal admittance matrix. The equation used by RELAX is:

$$J\cdot DELTAV=-f(V) \tag{2.24}$$

where J is the Jacobian of the function f(V) defined above.

$$DELTAV=-J^{-1}\, f(V) \tag{2.25}$$

This can reasonably be approximated by:

$$DELTAV=-\tilde{J}^{-1}\, f(V) \tag{2.26}$$

producing reasonable results using the approximate Jacobian $\tilde{J}$. When the formulation used by SPICE

is put into the above form, the result is:

$$\mathbf{J}\cdot\mathbf{V}_{new} = \mathbf{J}\cdot\mathbf{V}_{old} - \mathbf{f}(\mathbf{V}_{old}) \tag{2.27}$$

$$\mathbf{V}_{new} = \mathbf{J}^{-1}\left\{\mathbf{J}\cdot\mathbf{V}_{old} - \mathbf{f}(\mathbf{V}_{old})\right\} \tag{2.28}$$

Since G(V) is constructed to be identical to J, this becomes:

$$\mathbf{V}_{new} = \mathbf{J}^{-1}\left\{\mathbf{J}\cdot\mathbf{V}_{old} - \mathbf{J}\cdot\mathbf{V}_{old} + \mathbf{I}(\mathbf{V}_{old})\right\} \tag{2.29}$$

SPICE simplifies this as:

$$\mathbf{V}_{new} = \mathbf{J}^{-1}\mathbf{I}(\mathbf{V}_{old}) \tag{2.30}$$

Equation 2.28 can be simplified to:

$$\mathbf{V}_{new} = \mathbf{J}^{-1}\mathbf{J}\cdot\mathbf{V}_{old} - \mathbf{J}^{-1}\mathbf{f}(\mathbf{V}_{old}) \tag{2.31}$$

Attempting to eliminate the matrix inversion step and bypass the Jacobian evaluation results in:

$$\mathbf{V}_{new} = \tilde{\mathbf{J}}^{-1}\mathbf{J}\cdot\mathbf{V}_{old} - \tilde{\mathbf{J}}^{-1}\mathbf{f}(\mathbf{V}_{old}) \tag{2.32}$$

Unfortunately, this only produces a proper Newton iteration if $\tilde{\mathbf{J}}^{-1}\mathbf{J} = 1$, which is generally not true. Thus, Jacobian bypass is not suitable for use in a simulator such as SPICE while it computes V instead of ΔV. While it may be possible to convert SPICE to such a formulation, the benefits of such a major reformulation are not apparent. The programming cost is quite significant, requiring major changes throughout the program as well as a complete rework of all of the device models.

2.6. Faster Models

There have been many attempts to reduce the time required by SPICE to evaluate the various semiconductor devices. Some of these efforts have produced quite good results, but many of them approach the problem by reducing the accuracy or completeness of the model, thus requiring additional devices to fully implement real devices, or making the simulation much less realistic. As an example of this, examine the SPICE level one MOSFET model. This model has a relatively simple device equation giving I_d as a function of V_{gs} and V_{ds}. This part of the model requires about 45 lines of code and can be evaluated quickly. Unfortunately, this simple model isn't enough, so it must be embedded in a larger model that considers other effects. The additional effects which must be

considered even for the simple level one model are:

Source Resistance
Drain Resistance
Bulk-Source diode (with nonlinear capacitance)
Bulk-Drain diode (with nonlinear capacitance)
Nonlinear Gate-Source capacitance
Nonlinear Gate-Drain capacitance
Nonlinear Gate-Bulk capacitance

Looking at these components as well as the general overhead of the model during a long run, the time spent evaluating the total model can be summarized as:

Table 2.19
Level one MOSFET model evaluation time breakdown

Component	Percent of MOSFET time	time(ms/device-iteration)
Gate capacitance	26%	0.13
Diodes	19%	0.09
Matrix loading	16%	0.08
Overhead	12%	0.06
Bypass	9%	0.05
Evaluation	8%	0.04
Limiting	5%	0.03

As can be seen, some of this cost must be incurred regardless of the drain current model actually implemented: approximately 12% of the total model evaluation time being spent in various operations such as finding the starting terminal voltages and saving values for future iterations, and 16% spent adding the computed conductances and currents into the matrix. The 9% spend in bypass can be eliminated but at a cost of more time spent in each of the other categories, since on average each trip through the bypass code is much cheaper than the corresponding trip through the limiting, evaluation, diode, and gate capacitance code. Finally, the time spent in limiting can be reduced or moved, but a limiting scheme is still needed somewhere to damp out the wild swings sometimes encountered during Newton-Raphson startup or when moving rapidly along the exponential curve. When using the more sophisticated device models, all of the costs remain approximately unchanged except for the equation evaluation which rises to a worst case value of approximately 0.91 ms for the level two model. With the 0.45 ms overhead for all of the models, this permits a possible performance increase of approximately a factor of three for the worst case model before impacting the accuracy of the

supporting code and the associated models for effects other than drain current. Even if the model somehow includes the diodes and gate capacitance effects in its internal structure, the overhead will still be approximately 0.2 ms. This would allow a maximum possible performance increase of a factor of 7.5 for model evaluation and an overall simulation speedup of less than half that, significantly less than the 10 to 100 claimed by some from model speedup. Further speedups are certainly possible if the model is simplified. Simplifying the model by removing effects such as non-linear capacitances as in earlier versions of SPICE2, or removing the bulk terminal can have a larger effect. Such simplifications would change the cost of the overhead operations such as matrix loading and device limiting, as well as the cost of evaluating the drain current. These greatly simplified models, while possibly suitable for a logic or timing simulator, are not suitable for a circuit simulator such as SPICE.

CHAPTER 3

Convergence

SPICE uses a damped, iterative Newton-Raphson algorithm[Acto70a, Nage75a] to determine the solution to the circuit it is analyzing at any time. This algorithm is guaranteed to converge to the correct answer *if* it is started sufficiently close to that correct answer and is followed accurately. While SPICE attempts to ensure that it always starts "close enough" to the answer, it is not always possible to do so and the iteration may fail to converge. Various techniques to minimize this problem have been used in past versions of SPICE, and new techniques and improvements in these old techniques have been added to SPICE3.

A large part of the problem of convergence is properly choosing the tradeoff between convergence speed and robustness of the program. Convergence is fastest with simple linear models, but more accurate models for transistors introduce substantial non-linearities which must be modeled accurately for the result of the simulation to represent the circuit accurately. In addition, these non-linearities typically take longer to evaluate than simple linear models, so the choice of different models for different regions must be made carefully and the fitting of the regions together must also be performed carefully to ensure smooth transitions between regions without causing difficulties for the Newton-Raphson iteration.

3.1. Initial Guess: Failure to Converge in dc Analysis

The largest category of non-convergence problems that occurs in SPICE is that of failure to converge in the initial dc analysis. This analysis is the most error prone of the uses of the Newton-Raphson iteration method because there is, in general, no approximate solution available at low cost. The rate of convergence of the Newton-Raphson iteration is quadratic once the method gets "close enough" to the solution. Before reaching this point, the method generally converges at a slower rate, possibly even diverging. In the transient and dc transfer curve analyses, the solution at the previous

time or voltage point provides an approximation to the solution at the new point, but during the initial dc solution, no such information is readily available. The approximate solutions provided by the transient or transfer curve analysis is almost always "close enough" to lead to a rapid convergence.

In the case of the dc solution, the problem is the trade-off between the cost of generating a "close enough" guess and the cost of simply allowing the Newton-Raphson iteration process to proceed from a less accurate guess. Experience with SPICE2 has shown that in most cases, the solution is found reasonably quickly and inexpensively by simple letting the iteration proceed from a less accurate guess. The general strategy adopted for SPICE3 is to use this simple, efficient procedure initially and when it begins to break down (by failing to converge to a solution in a reasonable number of iterations), to use a more complex and potentially more time consuming algorithm to recover and ensure that a solution is found.

3.1.1. Previous approach

SPICE2 uses a very simple set of assumptions about the initial dc solution which frequently work, but fail often enough to be a serious concern. The initial assumption in SPICE2 is that all node voltages and source currents are zero, but that all device junctions are at their "critical" voltage (V_{t_0} for MOSFETs, $V_t \ln\left[\frac{V_t}{\sqrt{2}\times I_s}\right]$ for BJTs and diodes, -1 for JFETs and MESFETs). Unfortunately, the route from this guess to the correct operating point is frequently complex, with large amplification and feedback stages resulting in numerical overflow and/or oscillation problems. Further, this guess is not consistent, since a MOSFET will be assumed to have all terminals at 0 but, for example, 0.6v from gate to source and -1 volt from bulk to source. SPICE2 provides a variety of ways through the ".NODESET" and ".IC" statements for the user to point the program toward the correct initial solution, but these require considerable knowledge about the operation of the circuit to be provided by the user and make the situation even worse when used incorrectly.

3.1.2. Source propagation

Source propagation is a form of logic simulation that has been proposed as a technique for getting an approximate dc solution[Deva85a]. This technique, as implemented in CSIM, is most directly applicable to digital MOS circuits. The technique uses a switch-level steady-state logic simulation of the circuit to obtain starting estimates for node voltages. In this technique, the network is traversed from the input voltage sources. A technique similar to an event driven logic simulation is used to assign logic levels to nodes directly driven by the sources. The logic levels are propagated throughout the circuit if possible. Where the logic levels cannot be obtained by this process, reasonable off, on, and intermediate conductances based on W/L ratios are used for the MOSFETs with voltage sources representing appropriate values for each logic level driving the known nodes. This generates a network of voltage sources and resistors which is then solved, generating values for each node. These node voltages are then used as the initial guess for the dc solution of the circuit. Work conducted by Srinivas Devadas [Deva85a] on this subject show good results, not only making the operating point solution less likely to fail, but reducing the number of iterations required to reach that operating point by a factor of up to 10 as shown in Table 3.1. Unfortunately, while it may be possible to extend this technique to non-digital and non-MOS circuits, the size and complexity of the code required to implement source propagation makes it a poor choice for a general-purpose simulator such as SPICE3. Techniques such as this are appropriate for a more sophisticated user interface and front end to SPICE3 such as NECTAR[Kele88a], which may be able to recognize this particular situation or determine that it is the case through its interaction with the user.

3.1.3. Continuation Methods

Continuation methods[Orte70a] are one means of dealing with a major limitation of Newton-Raphson methods, namely that convergence to the solution is only guaranteed if the iteration is started from a point "close enough" to the solution. Continuation methods provide a means of obtaining a point "close enough" to the solution to allow the Newton-Raphson iteration to succeed. Continuation methods accomplish this by converting the task from the solution of a single problem to the solution

Circuit Description	Operation	SPICE2 CPU time(s)	CSIM CPU time(s)	Ratio
500 inverter chain	Readin	135.1	3.4	39.7
Matrix size 504	Setup	12.6	2.1	6.0
N devices 500	dc CPU/Iter	8.50	1.06	8.0
P devices 500	tran CPU/Iter	14.28	1.83	7.8
Total devices 1000	dc CPU/Iter-Device	0.00850	0.00106	8.0
	tran CPU/Iter-Device	0.0148	0.00183	8.5
	Total dc Iters	51	6	8.5
	Total dc CPU	433.5	6.4	67.7
Soar ALU	Readin	225.1	9.6	23.4
Matrix size 1183	Setup	78.8	4.1	19.2
N devices 1289	dc CPU/Iter	28.0	2.75	10.18
P devices 403	tran CPU/Iter	36.0	3.68	9.8
Total devices 1692	dc CPU/Iter-Device	0.01654	0.001625	10.18
	tran CPU/Iter-Device	0.02127	0.002174	9.8
	Total dc Iters	44	9	4.88
	Total dc CPU	1232.0	24.75 .	49.7

Table 3.1
Iteration reduction using CSIM
From *CSIM User's Manual and Report*

of a continuous set of problems. One member of the set of problems is easily computable, and the other members proceed continuously as a function of a single variable to the original problem. By varying this control variable slowly enough, the successive points taken are each close enough to the previous one for the Newton-Raphson iteration to converge. This gives us a way to approach the solution more gradually, thus making convergence much more likely, although with increased cost.

Although the techniques presented here are all continuation methods, they approach the problem from three distinct directions and have their own advantages and disadvantages.

3.1.3.1. G_{min} stepping

G_{min} stepping is a combined algorithm which comes from a variety of sources. The original term "G_{min}" comes from Nagel's[Nage75a] SPICE2 work where it referred originally to a small conductance from each node to ground. It was later used for a minimal junction conductance which was inserted to keep the matrix well conditioned. In BIASL.25[Newt76a] this conductance was stepped from

a large value to a much smaller one to aid convergence. In SPLICE3[Sale86a] and RELAX2[Webb86a] this conductance has been separated from the minimal junction conductances in the devices, which are not used; the node to ground conductances are stepped down as a dc convergence aid. SPICE3 uses both Nagel's technique of a constant minimal junction conductance to keep the matrix well conditioned and a separate variable conductance to ground at each node as a dc convergence aid. These variable conductances make the solution converge faster; they are then reduced and the solution re-computed. Eventually, the solution is found with a sufficiently small conductance. Finally the conductance is removed entirely to obtain a final solution, implementing a form of *continuation method*[Orte70a].

Implementing G_{min} stepping in SPICE3 is not too complicated if one additional step is taken and G_{min} is actually added to the diagonal of the reordered matrix, not the matrix corresponding to the original circuit. This has the advantages of not creating any new non-zero entries in the matrix as well as being simple to implement in the lowest level of the matrix package. This technique has produced very good results. Most of the circuits G_{min} stepping has been tested on have converged quite quickly, and SPICE3 now uses this technique by default when convergence problems occur.

There are many algorithms for picking the initial value of G_{min}, for reducing it, and for determining the last value to use before removing it from the circuit entirely. As an initial test, a simple

```
Gmin = Initial value;
while Gmin != 0 {
   itercount = 0;
   converged = false;
   while ( not converged ) {
      if ( itercount > iterlimit ) fail; /* don't loop forever */
      load matrix;
      solve circuit;
      converged = convergeTest();
      itercount = itercount + 1;
   }
   reduce Gmin;
}
```

Figure 3.2
G_{min} stepping algorithm.

logarithmic step is used by taking powers of 10 from 10^{10} times the minimal conductance (default 10^{-14}) down to that minimal conductance value from which point it is dropped to zero. This simple algorithm allowed all well-formed test circuits that did not generate numerical errors, except one*, to converge to an operating point. This degree of success using such a simple algorithm (and generally using less than the 100 iterations needed to trigger it) led to its being retained in this form. More elaborate stepping schemes are possible, but do not seem to be justified by the results obtained by the simple scheme.

3.1.3.2. Source stepping

Source stepping is another continuation method which begins by reducing the values of all the sources in the circuit to a small fraction of their value and obtaining a dc solution. Since the solution to the circuit with all sources at zero is known to be zero, when the source values are very close to zero, the solution should be very close to zero. By gradually increasing the sources, the answer should remain within the necessary range of the previous solution, thus guaranteeing convergence. Unfortunately, this technique doesn't work for all cases; in particular, when the input voltage reaches the level necessary to turn on transistors in digital circuits, the solution can change sufficiently rapidly to cause convergence failure. Since the code required to implement this technique is almost insignificant, it has been left in as a back-up to the more successful G_{min} stepping method described above.

3.1.3.3. Pseudo-transient

The pseudo-transient method of finding a dc solution is a variation on the source stepping method. In this method all sources are set to zero as in source stepping. [Week73a] The circuit is solved and the simulator then allows the sources to slowly ramp up to their nominal values as time progresses, gradually charging the capacitances of the circuit as it would if the circuit were actually powered up. Eventually, the sources reach their steady state value and the circuit voltages and

* The one circuit which did not converge is a fifteen stage CMOS ring oscillator, an extremely difficult circuit to solve.

```
srcfact = 0;
while ( srcfact != 1 ) {
        converged = 0;
        itercount = 0;
        while ( not converged ) {
                load matrix; /* voltage sources scaled by srcfact */
                increment itercount;
                solve circuit;
                converged = convergeTest();
        }
        increase srcfact;
} /* srcfact MUST be 1 by here */
```

Figure 3.3
Source stepping algorithm.

currents stop changing, indicating that the dc solution has been reached. This allows the capacitances and inductances of the circuit to damp out the oscillation frequently encountered during the early iterations of the dc analysis. This technique may prove useful at some time in the future, but has not been implemented at this time due to the tremendous success of the G_{min} stepping algorithm described above in handling all of the problems the pseudo-transient method addresses.

3.2. Transient Problems

The problem of convergence in transient simulation is a superset of the problem of convergence at dc. A transient analysis will generally be much more time consuming than a dc analysis, and thus every possible attempt must be made to reduce simulation time. Due to the nature of the problem, errors accumulate from timepoint to timepoint, thus making accuracy even more important. Furthermore, since time is now a variable, the effects of capacitance and inductance must be considered in addition to all of the normal dc effects.

3.2.1. Bypass

One of the features of SPICE2 which has received mixed reviews for many years is the inactive device bypass code. As described in Section 2.5, this code allows a device, the terminal conditions of

which have not changed significantly since its last evaluation, to have the results of the previous evaluation re-used instead of repeating the evaluation. Since the cost of device evaluation can be quite high, particularly for the more complex devices, and since many devices do not change for long periods of time, particularly in digital circuits, this technique can save a large amount of computational time.

Unfortunately, many users have had difficulties with the inactive device bypass in SPICE2 and have concluded that the problem lies with the technique. A careful analysis of the implementation of SPICE2 during the writing of SPICE3 and during subsequent debugging revealed that much of the bypass code used in recent versions of SPICE2 was incomplete. This created the potential to cause serious problems in large and complex circuits, although it is quite well behaved for small, single device circuits of the type typically used to test such code.

Consider, for example, the BJT. In SPICE2, the bypass code reloads a list of variables which does not include CAPCS, the collector to substrate capacitance, then branches directly to statement number 800, near the end of the BJT subroutine, where it passes the value of the uncomputed variable CAPCS to the subroutine INTGR8 to compute the equivalent conductance for the capacitor. Under certain conditions, CAPCS may have been saved, but since it isn't restored, an incorrect value will be used in its place. In FORTRAN, this actually works much of the time because FORTRAN local variables are *static*, thus CAPCS will have whatever value was last assigned to CAPCS in the previous iteration within BJT *even if in a previous call* to BJT. Thus, for a one transistor circuit, it will still have the correct value of CAPCS, regardless of what other subroutines have been called between the solutions. In the case of a more complicated circuit, it will have the value of CAPCS for the last bipolar transistor that was actually evaluated, regardless of the parameters of that device. Since many users group similar transistors together, and most devices in a circuit will have similar characteristics, this masks the problem further. In SPICE3, this problem is immediately apparent. In C, variables are dynamically allocated on the stack each time a subroutine is entered. Hence the use of the uninitialized variable CAPCS results in a random number from the stack being used. Thus the variable could

have a value from almost any other routine, including integers or bit fields, producing unpredictable results. These results frequently include floating point errors since the magnitude of the random numbers is unlikely to be correct for junction capacitances, and experience indicates that VAX reserved operands are not uncommon.

Fixing these problems in the various device models has improved the reliability of the bypass technique. In conjunction with other changes made in this project, this has fixed numerous "timestep too small" errors. For those users who still do not wish to employ this technique, SPICE3 also allows inactive device bypass to be disabled at either compile time or run time.

3.2.2. Discontinuities and Inconsistencies

Another major problem in finding the solution, one which affects both the initial dc and the transient solutions, is the consistency of the models used. Newton-Raphson can only converge if it is used correctly; by incorrectly evaluating the device models or the derivatives needed for the Jacobian matrix, the algorithm can be led astray, leading to non-convergence. Unfortunately, SPICE has traditionally had problems with this particular situation because of the difficulty of errors made in computing companion model values for complex models, or because changes have been made in one part of the code for a model without updating all other, related parts of the model code.

In this section, a number of such problems in SPICE are discussed, with applicability to both SPICE2 and SPICE3.

3.2.2.1. Diode models

SPICE models the source to bulk and drain to bulk junctions of MOSFETs with simple junction diodes, with the corresponding non-linear capacitance. These capacitances make up only a small part of the overall model, but are vital to the proper operation of many circuits. As described in Section 2.1.3.1, SPICE uses four different equations for capacitance and charge in a junction for the regions:

$$V_j \leq F_c P_b \tag{3.1}$$

and

$$V_j \geq F_c P_b \tag{3.2}$$

where V_j is the junction voltage, F_c is a curve fitting parameter between zero and one, and P_b is the bulk junction potential. For convergence, SPICE must also use the proper variable capacitance:

$$\frac{dQ_1}{dV_j} = C_1 \tag{3.3}$$

$$\frac{dQ_2}{dV_j} = C_2 \tag{3.4}$$

where the subscripts 1 and 2 indicate the two regions defined by Equations (3.1) and (3.2) respectively, with C and Q being the capacitance and charge on the junction capacitance. At the intersection of these regions $V_j = F_c P_b$ continuity requires that:

$$Q_1 = Q_2 \tag{3.5}$$

$$C_1 = C_2 \tag{3.6}$$

and

$$\frac{dC_1}{dV_j} = \frac{dC_2}{dV_j} \tag{3.7}$$

The actual equations used in SPICE2 for MOSFETs are:

$$C_1 = \frac{C_{j0}}{(1 - \frac{V_j}{P_b})^{M_j}} \tag{3.8}$$

$$Q_1 = \frac{C_{j0} P_b (1 - (1 - \frac{V_j}{P_b})^{1-M_j})}{1 - M_j} \tag{3.9}$$

$$C_2 = C_{j0}(1 - F_c)^{-M_j - 1}(\frac{M_j V_j}{P_b} - F_c(M_j + 1) + 1) \tag{3.10}$$

$$\begin{aligned} Q_2 = {} & C_{j0}(1 - F_c)^{-M_j - 1} M_j (V_j^2 - F_c^2 P_b^2) \\ & + C_{j0}(1 - F_c)^{-M_j - 1}(1 - F_c(M_j + 1))(V_j - F_c P_b) \\ & + \frac{C_{j0}(1 - (1 - F_c)^{1-M_j}) P_b}{1 - M_j} \end{aligned} \tag{3.11}$$

Where C_{j0} is the zero bias junction capacitance and M_j is the junction grading coefficient.

Checking these equations against the requirements of Equations (3.3) through (3.7),

$$\frac{dQ_1}{dV_j} = \frac{C_{j0}P_b}{1-M_j}(1-M_j)\frac{1}{(1-\frac{V_j}{P_b})^{M_j}}\frac{1}{P_b} \tag{3.12}$$

$$\frac{dQ_1}{dV_j} = \frac{C_{j0}}{(1-\frac{V_j}{P_b})^{M_j}} \tag{3.13}$$

$$\frac{dQ_2}{dV_j} = 2C_{j0}(1-F_c)^{-M_j-1}M_jV_j + C_{j0}(1-F_c)^{-M_j-1}(1-F_c(M_j+1)) \tag{3.14}$$

$$\frac{dQ_2}{dV_j} = C_{j0}(1-F_c)^{-M_j-1}(2M_jV_j - F_cM_j - F_c + 1) \tag{3.15}$$

$$\frac{dQ_2}{dV_j} \neq C_2 \tag{3.16}$$

at $V_j = F_cP_b$ this gives:

$$Q_1 = \frac{C_{j0}P_b(1-(1-F_c)^{1-M_j})}{1-M_j} \tag{3.17}$$

$$Q_2 = \frac{C_{j0}P_b(1-(1-F_c)^{1-M_j})}{1-M_j} \tag{3.18}$$

Thus:

$$Q_1 = Q_2 \tag{3.19}$$

And

$$C_1 = \frac{C_{j0}}{(1-F_c)^{M_j}} \tag{3.20}$$

$$C_2 = C_{j0}(1-F_c)^{-M_j-1}(1-F_c-M_jF_c+M_jF_c) \tag{3.21}$$

$$C_2 = \frac{C_{j0}}{(1-F_c)^{M_j}} \tag{3.22}$$

Thus:

$$C_1 = C_2 \tag{3.23}$$

Finally:

$$\frac{dC_1}{dV_j} = \frac{C_{j0} M_j (1 - \frac{V_j}{P_b})^{-M_j - 1}}{P_b} \tag{3.24}$$

$$\frac{dC_2}{dV_j} = \frac{C_{j0} (1 - F_c)^{-M_j - 1} M_j}{P_b} \tag{3.25}$$

Giving:

$$\frac{dC_1}{dV_j} = \frac{dC_2}{dV_j} \tag{3.26}$$

These all appear to be correct, except $Q_2 \neq \int C\, dv$, so that equation must be re-derived.

$$Q_2 = \int_0^{V_j} C\, dV \tag{3.27}$$

$$Q_2 = \int_0^{F_c P_b} C\, dV + \int_{F_c P_b}^{V_j} C\, dV \tag{3.28}$$

$$Q_2 = \frac{C_{j0}(1 - (1 - F_c)^{1 - M_j}) P_b}{1 - M_j} + \int_{F_c P_b}^{V_j} (C_{j0}(1 - F_c)^{-M_j - 1}(1 - F_c(1 + M_j) + \frac{V M_j}{P_b}))\, dV \tag{3.29}$$

$$Q_2 = \frac{C_{j0}(1 - (1 - F_c)^{1 - M_j}) P_b}{1 - M_j} + \left[C_{j0}(1 - F_c)^{-M_j - 1} (\frac{M_j V^2}{2P_b} - F_c(M_j + 1)V + V) \right]_{F_c P_b}^{V_j} \tag{3.30}$$

$$Q_2 = \frac{C_{j0}(1 - (1 - F_c)^{1 - M_j}) P_b}{1 - M_j} + C_{j0}(1 - F_c)^{-M_j - 1}(V_j - F_c P_b) - C_{j0}(1 - F_c)^{-M_j - 1} F_c (M_j + 1)(V_j - F_c P_b) + \frac{C_{j0}(1 - F_c)^{-M_j - 1} M_j (V_j^2 - F_c^2 P_b^2)}{2P_b} \tag{3.31}$$

$$Q_2 = +\frac{C_{j0}(1-(1-F_c)^{1-M_j})P_b}{1-M_j}$$
$$+C_{j0}(1-F_c)^{-M_j-1}(1-F_c(M_j+1))(V_j-F_cP_b) \quad (3.32)$$
$$+\frac{C_{j0}(1-F_c)^{-M_j-1}M_j(V_j^2-F_c^2P_b^2)}{2P_b}$$

at $V_j = F_cP_b$,

$$Q_2 = \frac{C_{j0}(1-(1-F_c)^{1-M_j})P_b}{1-M_j} \quad (3.33)$$

Thus:

$$Q_2 = Q_1 \quad (3.34)$$

$$\frac{dQ_2}{dV_j} = \frac{C_{j0}(1-F_c)^{-M_j-1}M_jV_j}{P_b} + C_{j0}(1-F_c)^{-M_j-1}(1-F_c(M_j+1)) \quad (3.35)$$

$$\frac{dQ_2}{dV_j} = C_2 \quad (3.36)$$

Equations 3.8 through 3.10 and Equation 3.32 provide the corrected junction capacitor model used in SPICE3. This set of equations is used for each of the four junctions used in the MOSFETs, and only models the capacitive effect of that junction. Finally, note that these equations are then modified as described in Section 2.1.3.1 for more efficient implementation, giving:

$$V = \left[1-\frac{V_j}{P_b}\right]^{-M_j} \quad (3.37)$$

$$C_1 = C_{j0} \times V1 \quad (3.38)$$

$$Q1 = \frac{C_{j0}P_b \times \left[1-V1 \times \left[1-\frac{V_j}{P_b}\right]\right]}{1-M_j} \quad (3.39)$$

$$C_2 = K1 + K2 \times V_j \quad (3.40)$$

$$Q2 = K3 + V_j \times \left[K1 + V_j \times \frac{K2}{2}\right] \quad (3.41)$$

For constants:

$$K1=C_{j0}\times(1-F_c\times(1+M_j))\times(1-F_c)^{-1-M_j} \tag{3.42}$$

$$K2=\frac{C_{j0}M_j}{P_b}\times(1-F_c)^{-1-M_j} \tag{3.43}$$

$$K3=\frac{C_{j0}P_b}{1-M_j}\times(1-(1-F_c)^{1-M_j})-\frac{K2}{2}\times F_c^2\times P_b^2-k1\times F_c\times P_b \tag{3.44}$$

While removing this discontinuity does not seem to solve any particular convergence or run time problems in SPICE, it does point out the general complexity of the models. In this case, somewhere along the way a divisor was dropped from one term in an equation, making two equations that *must* be consistent with one another inconsistent. This particular error occurred in a part of the curve where the device should never be operating, but can not be overlooked as a possible cause of trouble during the initial Newton-Raphson iterations. It also illustrates that not all errors in the code will show up as obvious errors in the simulation, and that no part of the code can be left unexamined in the search for errors of this type. In the next section a similar error with much more serious consequences is explored.

3.2.2.2. Meyer model

The Meyer model used in SPICE2 and early versions of SPICE3 is not the original Meyer model[Meye71a], but a modification made at some point in the past. Unfortunately, this modification, which was intended to include bulk voltage effects, causes the gate-drain and gate-source capacitances to be discontinuous when V_{ds} crosses zero.

The original capacitor model proposed by Meyer provided a single equation for the gate charge

$$Q_g=\frac{2}{3}C_{ox}\left[\frac{(V_{gd}-V_t)^3}{(V_{gd}-V_t)^2-(V_{gs}-V_t)^2}-\frac{(V_{gs}-V_t)^3}{(V_{gd}-V_t)^2-(V_{gs}-V_t)^2}\right] \tag{3.45}$$

This equation can be differentiated with respect to the gate-source and gate-drain voltages giving equations for the gate-source and gate-drain capacitances respectively.

$$C_{gs}=\frac{2}{3}C_{ox}\left[1-\frac{(V_{gd}-V_t)^2}{(V_{gs}-V_t+V_{gd}-V_t)^2}\right] \tag{3.46}$$

$$C_{gd}=\frac{2}{3}C_{ox}\left[1-\frac{(V_{gs}-V_t)^2}{(V_{gs}-V_t+V_{gd}-V_t)^2}\right] \tag{3.47}$$

Unfortunately, when these were modified to include a bulk voltage dependence, the capacitance terms were changed to:

$$C_{gd}=\frac{2}{3}C_{ox}\left[1-\frac{(V_{d_{sat}}-(V_{gs}-V_{gb}))^2}{(2V_{d_{sat}}-2(V_{gs}-V_{gb})-V_{db})^2}\right] \tag{3.48}$$

and

$$C_{gs}=\frac{2}{3}C_{ox}\left[1-\frac{(V_{d_{sat}}-(V_{gs}-V_{gb})-V_{db})^2}{(2V_{d_{sat}}-2(V_{gs}-V_{gb})-V_{db})^2}\right] \tag{3.49}$$

A problem occurs when the behavior of these terms is observed as V_{ds} is allowed to approach zero while keeping the device turned on and the bulk voltage far below the source and drain. At $V_{ds}=0$, these equations must be equal since, by the symmetry of the MOSFET, at that point the drain and source can be interchanged without affecting anything. Looking at these two equations at this point, and attempting to equate them, they are equal if and only if

$$V_{bs}=V_{ds} \tag{3.50}$$

Clearly this is not required of the MOSFET, so at the point where V_{ds} changes sign and SPICE reverses the source and drain to keep V_{ds} non-negative internally, a discontinuity is introduced into the capacitance. Graphically, this produces the capacitance curve shown in Figure 3.4.

This sudden change in capacitance will require an equally sudden change in either the charge on the capacitor or the voltage of the capacitor. If the voltage changes rapidly the truncation error tolerance will not be met. The sudden change in charge will call for a large current, which will also cause a truncation error problem, or a large voltage change elsewhere in the circuit. The result is a rejected timepoint, and a new much smaller timestep. This time, the same change in capacitance is present, and even less time to change the voltage or charge, so the currents involved will be even

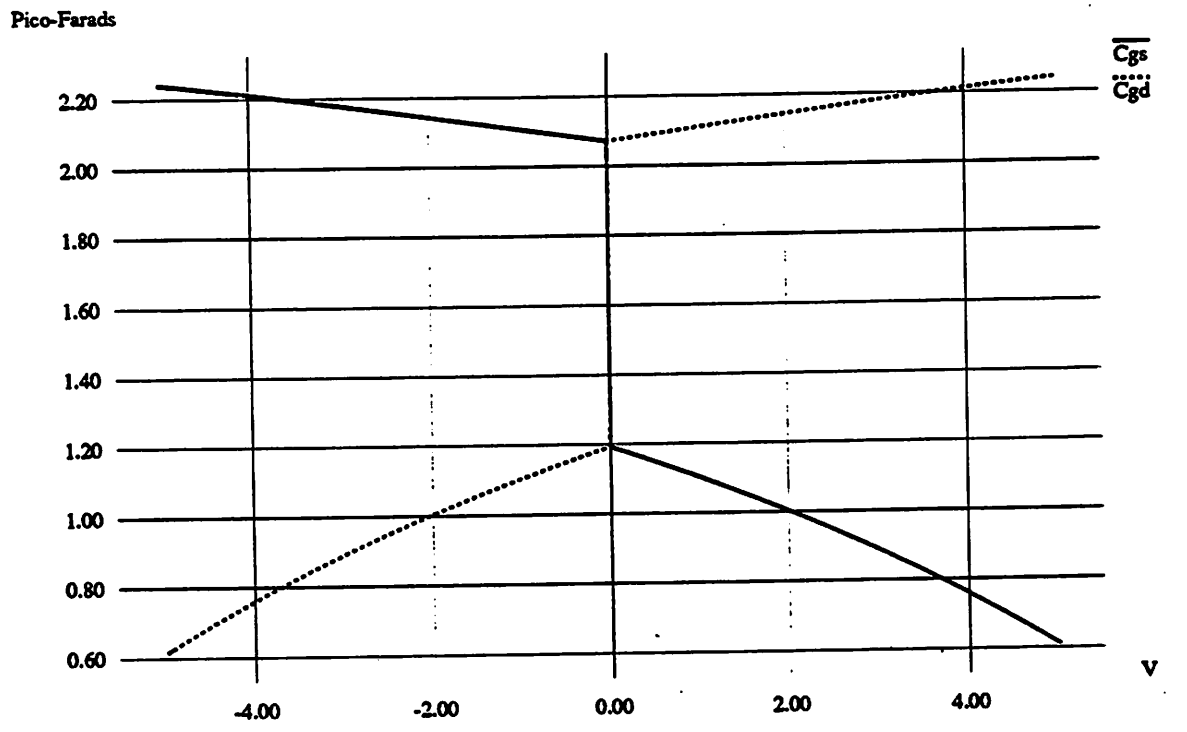

Figure 3.4
Gate-Source and Gate-Drain capacitance
vs Vds before correction

higher, eventually leading to a timestep too small error. Replacing this equation with the original Meyer equation produces the smoother curve of Figure 3.5.

This change in SPICE3 solves many of the "Timestep too small" problems previously encountered with both SPICE2 and SPICE3.

The development of an appropriate *continuous* bulk-voltage-dependent term remains as a problem for future work in model development.

3.2.2.3. Other MOSFET gate capacitance models

In addition to the convergence problems in the gate capacitance model used in SPICE, there have been numerous criticisms of the model used for its non-charge-conserving properties. The

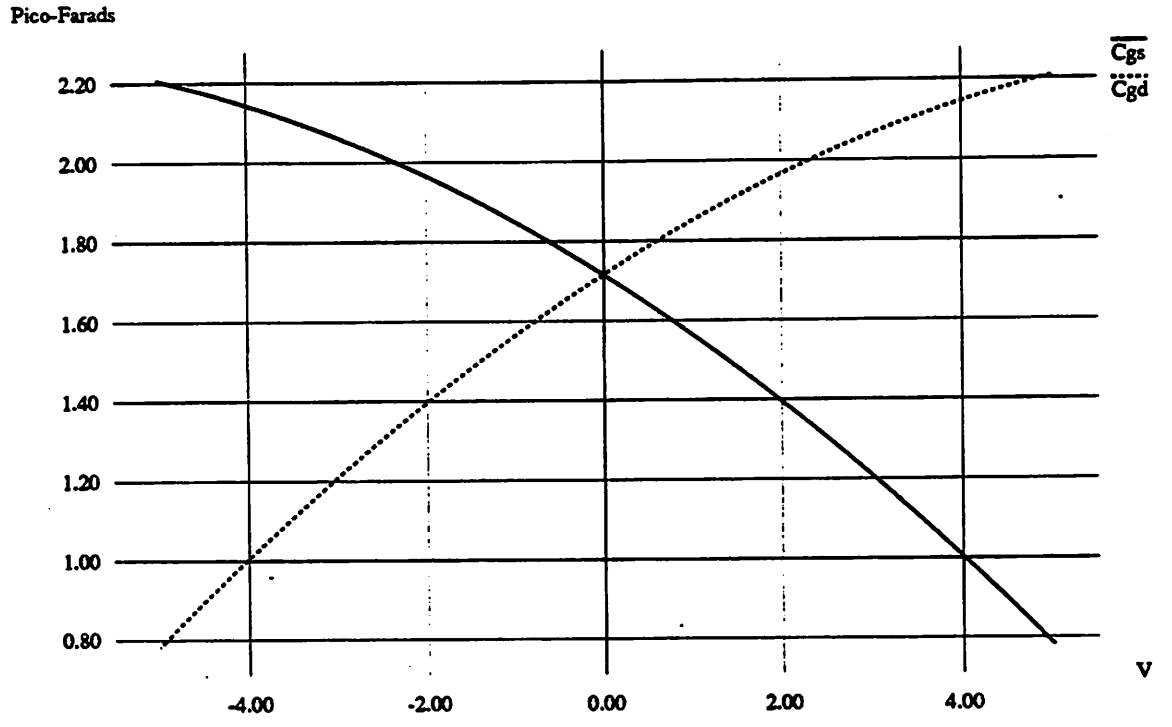

Figure 3.5
Gate-Source and Gate-Drain capacitance
vs Vds after correction

problem, as detailed by Sakallah, Yen, and Greenberg[Saka87a], is that the Meyer model is expressed as a single gate charge and all of the other charges are computed by numerically integrating the partial derivatives of that charge rather than by expressing those charges in closed form. Several solutions to this problem have been proposed.

Ward and Dutton[Ward78a] proposed a more complicated model which includes an explicit charge calculation for each terminal, thus producing a charge conserving model. This model was implemented in SPICE2 by Vladimirescu and Liu[Vlad80a] with some success. This implementation was restricted to the new level two MOSFET model, and added quite significantly to the complexity of that code.

Another model was proposed by Yang, and Chatterjee, [Yang82a] and Epler[Yang83a] which is also charge conserving. This model is also expressed as a set of charge equations for the terminals, with the need to compute all of the appropriate partial derivatives to obtain capacitances. Unfortunately, this model provides a full set of equations for various regions, but does not guarantee that these equations are consistent across the boundaries of these regions. As with the modified Meyer model in SPICE2, the Yang-Chatterjee model has discontinuities in the capacitance. Specifically, at $V_{ds}=0$ and $V_{gs}=V_t$, is the boundary between the linear and saturation regions. In these regions, the relevant equations are:

$$Q_{g_{linear}}=C_{ox}lw\left[\frac{\alpha_x V_{ds}^2}{12\left[-V_t+V_{gs}-\frac{\alpha_x V_{ds}}{2}\right]}+V_{gs}-V_{FB}-\frac{V_{ds}}{2}-\phi\right] \tag{3.51}$$

and

$$Q_{g_{saturation}}=C_{ox}lw\left[-\frac{V_{gs}-V_t}{2\alpha_x}+V_{gs}-V_{FB}-\phi\right] \tag{3.52}$$

Differentiating these to obtain C_{gs}, yields:

$$C_{gs_{linear}}=\frac{\partial Q_g}{\partial V_{gs}}=C_{ox}lw\left[1-\frac{\alpha_x V_{ds}^2}{12\left[-V_t+V_{gs}-\frac{\alpha_x V_{ds}}{2}\right]^2}\right] \tag{3.53}$$

$$C_{gs_{saturation}}=\frac{\partial Q_g}{\partial V_{gs}}=C_{ox}lw\left[1-\frac{1}{2\alpha_x}\right] \tag{3.54}$$

evaluating these two derivatives at $V_{ds}=0$ and as V_{gs} approaches V_t produces:

$$C_{gs_{linear}}=C_{ox}lw \tag{3.55}$$

and

$$C_{gs_{saturation}}=C_{ox}lw\left[1-\frac{1}{2\alpha_x}\right] \tag{3.56}$$

Since equations 3.54 and 3.55 are different for any non-degenerate case, there is a mismatch in the capacitance similar to the one observed to cause so many troubles in the Meyer model, and at the same point.

While SPICE2 has used the Ward-Dutton model and other simulators such as RELAX and SPLICE2 have used the Yang-Chatterjee model, at this time the standard Meyer model has been preserved in SPICE3 because of its simplicity and ease of evaluation. As the MOS models are evaluated further with the new flexibility of modifying the structures of the program, the model should probably be replaced by one of these two charge conserving models or the Sakallah, Yen, Greenberg charge conserving model. The choice of which one is implemented should be based on the complexity of the drain current model being used.

3.3. General Problems

There are several problems which apply equally to the dc and transient solutions of the circuit. Some of these problems can be addressed by the simulator relatively easily, such as problems in the matrix solution due to the circuit structure, others can not be solved that simply.

A major problem is that of circuits which have no unique solution. It is not uncommon to find circuits, particularly those generated by circuit extractors, which have floating nodes with no dc path to ground. These circuits *do not have a unique solution.* It is also common to find circuits that are not well-formed, particularly those which are incorrectly connected (e.g. the bulk of a MOSFET tied to the wrong rail). SPICE2 attempted to catch these and other, similar problems by a topological check performed before the simulation. No suitable topological check has been implemented in SPICE3, but something like this to perform preliminary checks is clearly needed. A program such as NECTAR[Kele88a] provides many of these facilities, allowing more complete checking than the topological check of SPICE2 could perform. Nectar achieves this through the addition of user interaction and the use of additional heuristics to look for common cases that, while not always an error or technologically incorrect, are usually an indication of a problem. Thus, including a sophisticated knowledge-based system like NECTAR in SPICE3 in the future, or the use of such a front end to examine SPICE3 inputs and pre-condition them before actually simulating them is advisable.

3.3.1. Floating nodes

One characteristic that has been blamed for many SPICE convergence problems in the past is "floating nodes" . These are circuit nodes which have no conductance from the node to any other part of the circuit. This produces a row and column of zeros in the modified nodal analysis matrix, making it impossible to invert the matrix. In a very high quality memory circuit, this is the ideal for a memory cell but cannot be realized in practice. Most of the problems caused by "floating nodes" in a circuit have, in fact, been either input errors or discontinuities in capacitance at nodes with very small conductances connected to them. This leads to numerical errors. Attempts to isolate circuits with real floating nodes have always led to another problem in SPICE being found that, when fixed, corrected the problem. This particular problem is very common in CMOS circuits, where nodes are frequently cut off. In this case the capacitance discontinuity in the Meyer model described in Section 3.2.2.2 exhibits itself quite frequently, since both source and drain are frequently at a potential different from bulk. Since no non-contrived examples of floating node circuits were left to work from, this "problem" remains unsolved.

3.3.2. Matrix pre-ordering

There are special situations which must be handled in the matrix package of a program which performs modified nodal analysis(MNA)[Nage75a, Cohe76a, Yang80a, Ho75a]. While these situations do not keep the program from computing correct answers, they do make it more difficult. The difficulty is caused by elements such as voltage sources which produce structural zeros on the diagonal of the matrix (Figures 3.6 and 3.7).

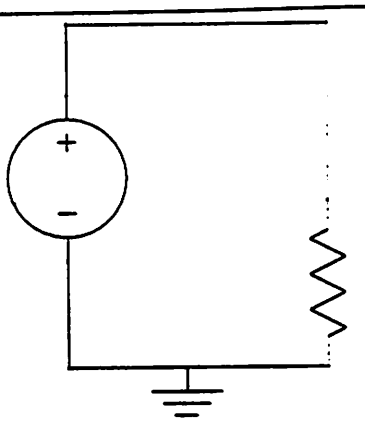

Figure 3.6
Voltage source causing MNA problems

$$\begin{bmatrix} G & G & 1 \\ G & G & 1 \\ 1 & 1 & \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_V \end{bmatrix} = \begin{bmatrix} \\ \\ V \end{bmatrix}$$

Figure 3.7
MNA matrix corresponding to Figure 3.6

Where these elements occur in isolation, the reordering algorithm handles them quite easily by noticing that the current equation with the zero on the diagonal can be either ignored, allowing it to be filled in by the reordering, or swapped with one of the node equations corresponding to the nodes the source is connected to, producing a suitable diagonal entry for both equations (Figure 3.8).

$$\begin{bmatrix} G & G & -1 \\ -1 & 1 & \\ -G & G & 1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_V \end{bmatrix} = \begin{bmatrix} \\ V \\ \\ \end{bmatrix}$$

Figure 3.8
MNA matrix corresponding to Figure 3.6 after row swapping.

In the case of the independent voltage source, this results in ones on the diagonal in both rows, clearly a desirable result in most MNA matrices.

As previously shown by others[Yang80a, Kund85a], the problem becomes apparent when two or more of these sources occur interconnected as in Figure 3.9.

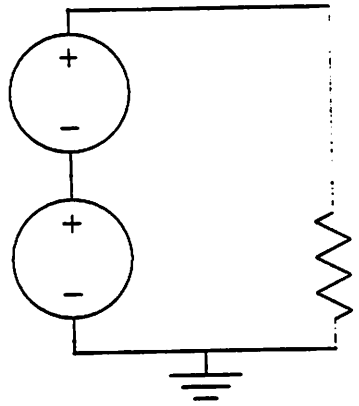

Figure 3.9
Circuit with multiple sources

In this case, the structural zeros marked † in Figure 3.10 must be cleared by swapping with other rows.

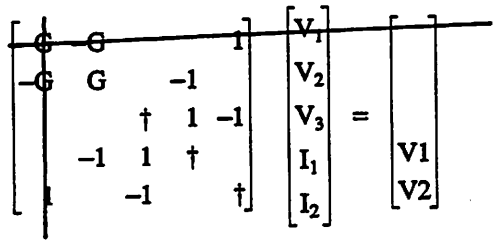

Figure 3.10
Matrix with poor numbering choice

This can lead to a simple solution if the sources are processed in the correct order, but an unlucky choice of numbering and thus ordering of the row swaps, can cause the situation shown in Figure 3.11 where the equation with the zero marked † must be swapped with a particular row which has *already* participated in a swap, and thus is ineligible to be swapped again as it will just put another zero on the diagonal.

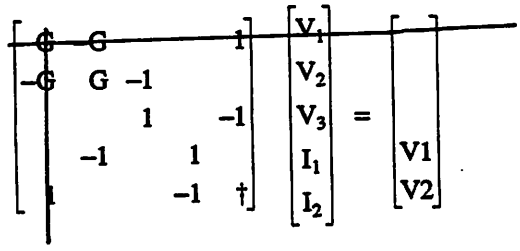

Figure 3.11
Matrix with poor numbering choice

In SPICE2[Nage75a] and YSPICE[Yang80a], this problem is solved by having special case code which iterates through the voltage sources and inductors to perform the proper row swapping. This is not acceptable in SPICE3 because it requires knowledge of the devices in the matrix handling code. To

solve this, a new algorithm has been developed which performs the appropriate row swaps without reference to the circuit simply by analyzing the matrix. This algorithm handles arbitrary interconnection of sources and connections to ground provided the MNA restriction of no loops of sources is adhered to.

The algorithm, as shown in Figure 3.12, begins by finding those structures that look like sources or inductors which are already constrained by having one end attached to a non-swappable row. Equation numbers corresponding to these structures are all pushed onto a queue. As the queue is processed, each equation is examined to check that it still has one free node equation to swap with, and that swap is performed. If no free node equation is available, there is a loop of source structures. If the equation swap is successful, the swapped equations are marked to prevent their participation in any future swaps, and are then examined to see if they are attached to any source structures. If any source structures are attached to the swapped equation, they are now constrained and must be added to the queue. When the queue is empty, any single equation corresponding to a zero on the diagonal may be pushed into the queue.

This algorithm will find and perform a comparable set of row swaps to the algorithms in SPICE2 and that given by Ping Yang[Yang80a], but operates without explicit knowledge of the sources in the circuit by exploiting the structure of the MNA matrix.

```
j=0;
toswap=0;
for ( i in 1 .. numeqn ) {
   if ( matrix_ii does not exist ) {
                increment toswap;
                status[i] = NEEDSWAP;
                count[i] = number of non-zero entries in row i;
                if ( count[i] == 1 {
                        status[i] = INQUEUE
                        push i into work queue;
                }
        } else {
                status[i] = OK;
        }
}
while ( toswap > 0 ) {
        while ( work queue is not empty ) {
                j = pop(work queue);
                k = 0;
find:
                find first non-zero element matrix_jk after current matrix_jk;
                if ( status[k] == LOCKED ) goto find;
                swap row j and row k;
                status[j] = LOCKED;
                status[k] = LOCKED;
                decrement toswap;
                for ( i in row j and matrix_ji != 0 ) {
                        decrement count[i];
                        if ( count[i] == 1 and status[i] == NEEDSWAP ) {
                                push i into work queue;
                                status[i] == INQUEUE;
                        }
                }
        }
        pick any i such that status[i] == NEEDSWAP;
        push i into work queue;
        status[i] == INQUEUE;
}
```

Figure 3.12
Matrix pre-ordering algorithm

CHAPTER 4

Program Architecture

A common problem in attempting to understand and modify a program such as SPICE is the sheer size of the program and the complexity of the program organization. Many people have found a need to modify SPICE2 for their own purposes by adding capabilities to it or modifying existing capabilities. This is very difficult to do in SPICE2 because of the organization of the program. For example, to add a new model to SPICE2, changes must be made in 25 different subroutines in addition to the writing of the model code itself. Many of the required changes are quite subtle and easy to overlook or perform incorrectly. In SPICE3, these various actions have been separated from the rest of the code. Up to 22 functions, most of them less than a page, may be required, but the requirements for each of these functions is fully specified and many can be omitted for simpler devices. Finally, a complete description of the device is built in a single file, including the names of whichever of the 22 possible functions have been implemented for the device. From this information, SPICE3 integrates the device into its operation as required.

To understand the structure of a large program such as SPICE3, it is necessary to break it into modules, then study them individually, and finally observe how they interact with each other to produce the overall result. In the case of SPICE3, the structure must be examined from three perspectives and at several different levels.

Because of the ever-changing field of circuit design and with new technology constantly requiring changes in simulation technology, it is most important that the simulator be as flexible as possible, consistent with high performance. This requires the ability to add new devices, new types of simulation, new analysis algorithms, and new ways of inputting and outputting data. To this end, SPICE3 has been made modular, making it possible to do these things with as little additional work as possible. This necessitates a number of descriptive data structures which describe capabilities at one level

of the code to other levels and other structures which store data for one level to hide it from other levels. These structures are frequently only distantly related and thus are described independently. Finally, it is necessary to understand the subroutine calling hierarchy and the way the modules call each other.

The basic calling structure of the program is as shown in Figure 4.1.

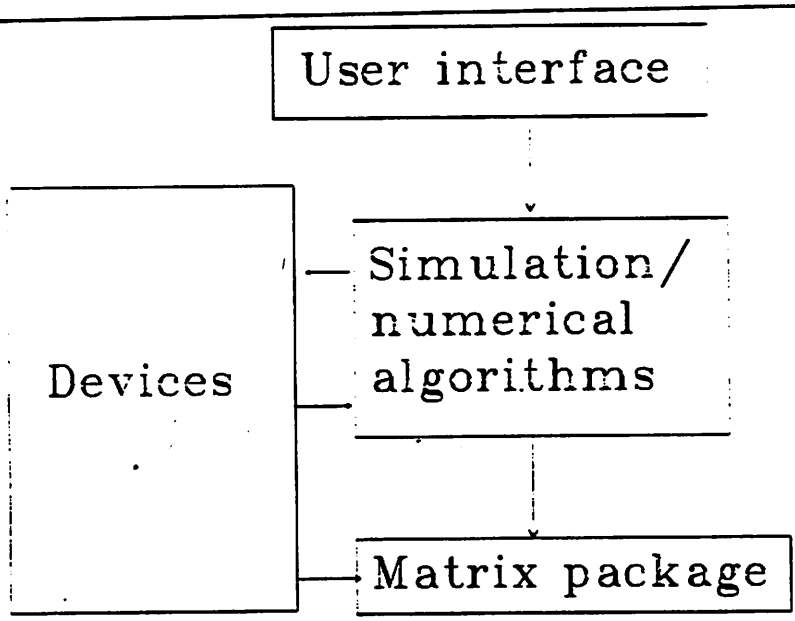

Figure 4.1
Basic calling structure

In this figure, the block labeled "Devices" represents all of the per-device-type packages which are incorporated into the program. These packages use and are used by the numerical algorithms of SPICE3. Both the device code and the numerical routines manipulate the sparse matrix through the matrix package. The simulation and numerical algorithms have been intermingled, since they are so closely related and have such a great effect on each other. The user interface only knows of the entry points provided by the simulation control routines, thus it only calls them and they control almost all of the non-error output.

The basic descriptive data structure is shown in Figure 4.2.

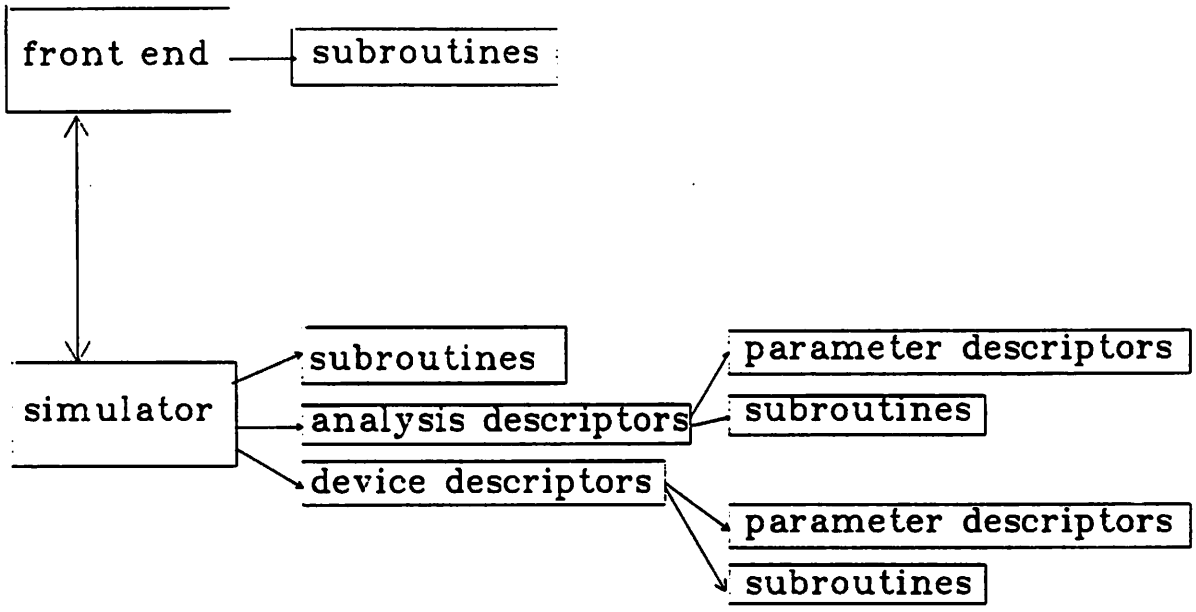

Figure 4.2
Descriptive data structure design

In the descriptive data structure, each of the minor packages exports a structure which contains a description of the capabilities of that package in a standard form so that the structures for all similar packages, either all of the device models or the analyses, can be collected into a single array at the next level up. At this minor package level, the description consists of one or more arrays of parameter descriptors and a set of pointers to functions which implement parts of the package's capabilities. At the next level, information on all of the packages in the simulator which will be needed by the front end are collected into a single structure and exported. Similarly, the front end collects all of the function pointers that will be needed by the simulator into a single data structure which it exports to the simulator during initialization. At run time, the front end and simulator each use the descriptor provided by the other to determine the capabilities available, the parameters used for various calls, and the types of the arguments called for by parameters.

Finally, the data hiding structure is as shown in Figure 4.3.

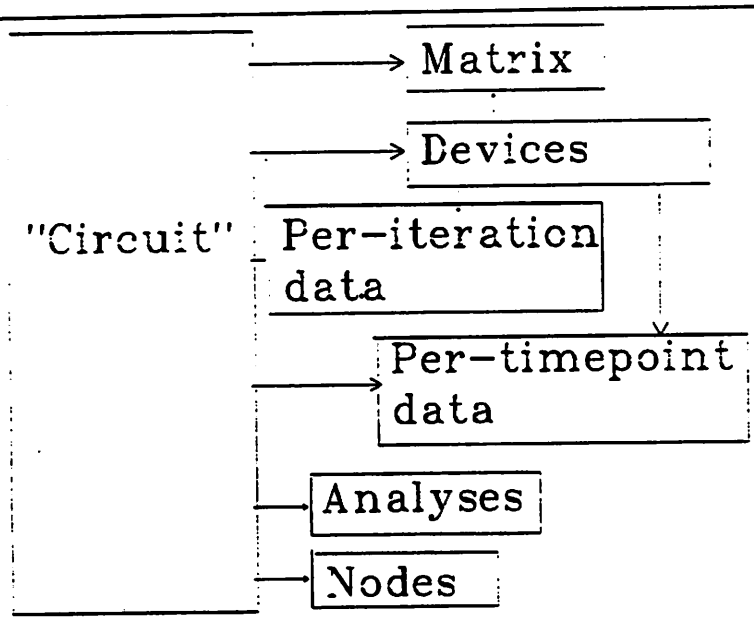

Figure 4.3
Local data hiding structure

In this structure, the "Circuit" structure is used to encapsulate all of the data related to, from the point of view of SPICE3, a single circuit. This may actually be a subcircuit of some larger simulator that uses SPICE3 as a subcircuit evaluator, but SPICE3 treats it internally as an independent circuit. This structure contains a considerable amount of global data, as well as pointers to more specialized structures for individual packages. The other structures are used to hold data that is either private to a single package, or must be indirectly referenced through a pointer. The matrix structure is completely private to the sparse matrix package, although other packages will obtain pointers to specific locations in the sparse matrix. The devices have a more complicated data structure described in detail below. The data that is renamed each iteration, the right hand side and solution vectors, is kept in a pair of dynamically allocated arrays which are referenced by much of the program. The data describing each analysis is maintained in a linked list of analysis-type-dependent structures for the private use of the code implementing each type of analysis. Finally, the descriptions of the nodes in the circuit is kept in a node table and maintained by a small number of routines within the circuit package itself.

The overall system architecture of SPICE3 is relatively simple. The program is broken into a group of independent blocks or packages, each of which only knows of the interface presented by the other packages and the operations they perform, not the details of their operations or data structures. This allows the packages to be maintained, enhanced, or replaced independently. The key to understanding the operation and structure of SPICE3 as a whole is an overview of these packages and their corresponding data structures as well as their interaction. The details of these interfaces are readily found in the documentation and are frequently apparent from the code using the package. The most important thing is to understand enough to know which package to look at when a given functionality is needed, or an error is found which appears to be associated with a particular type of operation.

4.1. Major Data Structures

The first step is to understand the basic data structures and their scope. In many ways, the program structure follows the data structure, and it is most easily understood by following the data structures.

The data area of the program is broken into three categories:

- Static data that is used to describe modules of the simulator to other modules of the simulator, the simulator to the front end, and certain physical constants. This area is pre-allocated and initialized in various parts of the simulator and stored as global static variables. All truly global data is *constant.*

- Other data that might normally be global data in a simulator. There is no variable global data in SPICE3. Widely used data has been packaged into a data structure, a pointer to which is passed to almost every routine to allow multiple instances of it to exist independently. This facilitates the use of the program on multiple circuits or as a subcircuit analyzer within a larger system. This data structure is the CKTcircuit structure, and most routines within the simulator which know any details of circuit simulation have a knowledge of this structure to use common "global" values.

- Finally, each package or group of routines may have its own private structure which it can define without affecting any other code.

Some packages, such as the numerical operations package, simply keep their data in the CKTcircuit structure; others have their own private structure which they allocate one or more instances of. Additionally, some packages are just instances of a *class* of similar packages and have structures which must comply with rules for that class of packages. The following sections provide an overview of the structures used throughout SPICE3. Further details of the structures needed to make modifications is provided in an appendix on data structure details.

4.1.1. Sparse matrix structures

The sparse matrix system uses two structures to represent matrices. The first structure, SMPmatrix, is used to provide an overall description of a matrix and is the primary structure. A single instance of this structure is used to represent an entire sparse matrix. This structure contains the basic framework of the matrix, but not the actual values of the elements or the details of the internal zero/non-zero structure. Pointers to the first non-zero entry in each row and column, as well as mapping data to convert between the row and column numbers used internally and those used by the calling program are contained in this structure to provide reasonably fast access to any part of the matrix.

The second structure, SMPelement, is used to represent the non-zero terms in the matrix. Each row and column pair which has ever had a non-zero value will have an instance of this structure allocated to describe it and contain its value. All SMPelement structures representing entries in the same row are linked together in ascending column number order, and all SMPelement structures representing entries in the same column are linked together in ascending row number order. The actual value of the matrix entry is stored in this structure rather than in an external array as in SPICE2.

4.1.2. Analysis structures

Analyses form a class of operations which can be treated identically by the other levels of the system. There are two data structures associated with each type of analysis supported by SPICE3. These structures are members of a class of similar structures common to all analysis types.

4.1.2.1. SPICEanalysis structure

The SPICEanalysis structure is the static descriptive structure which describes the analysis itself to the next level of the simulator which manages the overall simulation task without knowledge of any particular type of simulation. The IFanalysis public structure required by the front end interface specification is also included as a prefix of the SPICEanalysis structure. This structure is the only information about the analysis available to the other levels of the simulator except for the main simulator setup and analysis call made in CKTdoJob. This knowledge in CKTdoJob should be moved at a future time, but at this point is most conveniently kept embedded in one subroutine.

4.1.2.2. JOB structure

Each analysis to be performed by SPICE3 is represented by a structure in a linked list of "jobs" to perform. The header on each job in the linked list is identical and identifies the analysis type, name, and next analysis in the list. The remainder of the structure is analysis specific and stores the analysis specific variable data.

4.1.3. Device structures

Devices in SPICE3 form another class of objects which have a standardized interface. Each device type is represented by three structures describing its needs and capabilities. The entire class of structures has a set of properties in common which allow some operations to be performed on devices without knowing the type of device. Additional details of the structure are left to the device implementation and thus vary widely between devices. For each device type, there is a "two dimensional" data structure as illustrated in Figure 4.4 which contains all the information on all the models and instances. This structure consists of a linked list of models of the device type, each of which contains

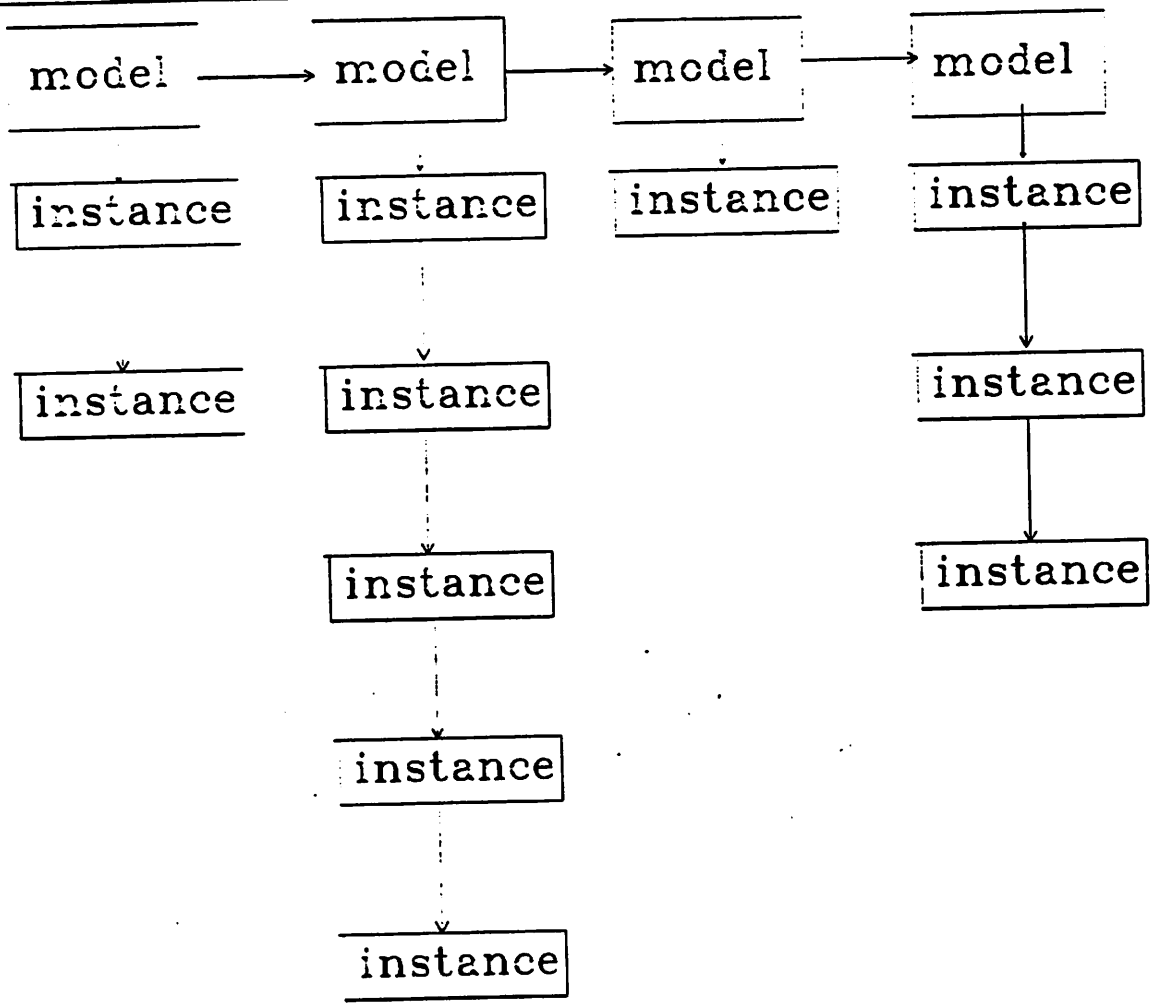

Figure 4.4
Model-instance data structure

a linked list of instances of that specific model.

4.1.3.1. DEVmodel structure

This structure contains all of the per-model variable data. An instance of this structure is allocated for each model of each device type in the circuit. Most of the structure is specified by the device implementor according to the guidelines in the chapter on adding devices to SPICE3, but a standard prefix is required on each structure that allows generic code to traverse the linked lists of models

to locate specific models and instances as needed. This prefix contains the linked list pointers, the model name, and the model type.

4.1.3.2. DEVinstance structure

This structure contains all of the per-instance variable data. Each instance of a model in the circuit will be represented by an instance structure. This structure contains the basic linkage information required to link it into the standard data structure as well as the instance name, a back pointer to the model, and any additional instance specific data needed for the particular type of device as determined by the implementor of that device.

4.1.3.3. SPICEdev structure

This is a static structure used to describe the device. One instance of this structure is allocated for each device type defined by SPICE3. This structure contains pointers to all of the functions used to implement the device and, as a prefix, contains the IFdevice structure required for the front end interface. This structure is the only data about the device available to the rest of the program.

4.1.4. Interface structures

These structures are used by the simulator to communicate with the front end and back end. These structures are not necessarily the optimum structures for SPICE3, but are designed to be general and allow for future extensions to SPICE3 as well as to support other simulators. Details of these structures can be found in the front end to simulator interface specification.

4.1.4.1. IFparm structure

The IFparm structure is used by the simulator to describe parameters of one of its objects to the front end. These structures generally appear as arrays with an accompanying count indicating the array length, with each IFparm structure describing a single parameter.

4.1.4.2. IFuid

IFuid is a pointer type used to described unique names to SPICE3. SPICE3 is not interested in the actual names of the various objects which are manipulated, but rather in the correct comparison of them. The IFuid type is a simple pointer type which SPICE3 can compare with IFuid's of other objects of the same type instead of having to compare the actual character strings or other encoding of the name of the object.

4.1.4.3. IFparseTree structure

The front end breaks an expression down into an IFparseTree structure as it is parsed. The structure contains a list of the variables the expression depends on as well as the information the front end will later need to evaluate the expression and its derivatives.

4.1.4.4. IFvalue structure

This is a simple union of many different data types used to pass any value between the front end and the simulator, including arrays of simple types.

4.1.4.5. IFnode structure

This is a structure used to pass node information back and forth between the front end and SPICE3. SPICE3 casts this to a CKTnode structure internally. For the purposes of the interface, IFnode is used so that both front end and simulator can refer to it.

4.1.4.6. IFcomplex structure

The IFcomplex structure is used by the front end to hold a complex number in x+y*i* format.

4.1.4.7. IFdevice structure

This is a structure used to describe a device and its parameters to the front end. The major feature of this structure is two arrays of IFparm structures describing the model and instance parameters of the device. It also contains much additional information about the device such as the number

of terminals the device has, the names of the terminals, the name of the device, etc.

4.1.4.8. IFanalysis structure

The IFanalysis structure provides information about the parameters available for a given type of analysis in much the same way as the IFdevice structure provides information about a device.

4.1.4.9. IFfrontEnd structure

The IFfrontEnd structure contains the pointers to all of the front end functions the simulator may need to call for output, error handling, and timing.

4.1.4.10. IFsimulator structure

This is a large structure used to describe SPICE3 to the front end. The IFsimulator structure contains pointers to arrays of several of the structures described above to convey complete descriptions of the capabilities of the program to the front end, as well as pointers to the functions in the front end that the simulator needs to call. This structure includes arrays of IFanalysis and IFdevice structures to describe the analysis and device capabilities to the front end.

4.1.5. CKTcircuit structure

This is the primary data structure of SPICE3. All variable data related to the description and operation of the circuit can be found within this structure, its substructures, or through pointers from this structure.

4.2. Control Flow

The control flow of the simulator is quite similar to the variable data structure. The front end performs parsing and user interface functions. Using the front end to simulator interface specification, the front end passes the description of the circuit and the desired analyses to the circuit package. In many cases the information is specific to a particular device or analysis, so after finding the proper data structures using the common prefixes, the device-specific or analysis-specific routine is called to

perform the actual operation. When the actual analysis is called for, the circuit package calls the device specific packages to perform all the necessary set-up operations as well as to initialize the sparse matrix for the simulation. The device specific packages then use the sparse matrix package to collect pointers to specific matrix locations they will be referencing regularly. Finally, the numerical iteration package takes is called, and it steps through the Newton-Raphson iteration at each solution point, using the device specific and matrix routines to perform the lower level operations.

4.3. Major Packages

The major packages used in SPICE3 can be split into two major categories, those which are specific and have only one instance, and those which are generic and have many instances, all of which follow the same basic framework.

4.3.1. Sparse Matrix Package

This package handles sparse matrices of the type encountered in circuit simulation situations. This is not a general package, but is tailored and tuned for the specific application. The package allocates matrices, creates elements in them, performs heuristic reordering based on knowledge of the structure of circuit simulation matrices, performs L-U factorization with and without reordering, and performs the forward-back substitution required to complete the solution of the circuit equations.

4.3.2. Circuit handling package

This set of routines handles the control of the simulation system as a whole. It guides the sequencing of analyses and the loops through the various devices, and provides an interface to the front end to allow it to access the data structures of the simulator.

4.3.3. Device packages

Each device is represented by a package which can perform all necessary actions for the simulator on instances and models of that type. This package provides a standard interface to the simulator, thus allowing it to deal with all devices in a uniform way. The types of actions available to the dev-

ice must be a superset of those needed by any device to be implemented in SPICE3, thus the package will have unused entry points for almost all device types.

4.3.4. Analysis packages

For each type of analysis to be performed by SPICE3, there will be a package which sequences through the basic algorithm of the analysis, calling on device functions, numerical package operations, matrix operations, and output operations as needed to perform the required analysis.

4.3.5. Numerical package

This is a package containing the basic numerical iteration and integration routines used by SPICE3. These compute integration coefficients for both predictors and correctors, as well as performing the actual prediction and driving the corrector iteration loop.

CHAPTER 5

Results

Ultimately, the most important measure of the quality of a program such as SPICE3 is its performance in terms of time, accuracy, and convergence on real circuits and in comparison with other simulators. Extensive tests on benchmark circuits have been run to evaluate the various improvements and techniques described in this report.

As a general-purpose simulator, SPICE3 must do well in direct comparison with similar simulators such as versions of SPICE2 from Berkeley and as improved through industrial use. Comparisons with special-purpose simulators such as RELAX[Whit85a] and SPLICE [Sale87a] are also important, but such comparisons must take into account the different capabilities, device models, and design objectives of the respective simulators.

5.1. About the Benchmarks

The benchmark circuits presented in detail in Appendix G have been used to compare SPICE3 with SPICE2 and, with some modifications and with translation to the appropriate input format, with RELAX running as a direct-method simulator*. Not all of these circuits are examples which can be run by SPICE2, or even by SPICE3. All circuits used for comparison and testing purposes have been added to this set of benchmarks, not just those which show particularly good results, thus there are still some circuits which do not converge when run with SPICE3. The set of circuits can be divided into several categories: those which SPICE2 and SPICE3 both solve, those which only SPICE3 can solve, and those which neither can solve. Of those which SPICE3 can solve and SPICE2 cannot, they can be further divided into those which SPICE2 cannot solve because the circuits require devices which SPICE2 does not implement and those which SPICE2 simply fails to solve. Finally, there are

* Similar comparisons could also be made with SPLICE running in direct mode, but under such conditions the basic algorithms embodied in SPLICE and RELAX are almost identical.

two versions of some circuits for which SPICE3 requires a modified input format for a device or has added parameters which improve performance, but they are listed as only a single circuit in the comparisons.

In the following sections, basic performance results are presented for each of the circuits in these categories. Unless noted otherwise, all of the results presented are from runs performed on a Digital Equipment Corporation VAX 8650 computer running the Ultrix 3.0 operating system. All of the raw data for these comparisons is tabulated in Appendix G, along with other statistics for these circuits. Plots are presented with the "circuit number" , which represents an arbitrary ordering of the circuits equivalent to the ordering of the tables, on the X axis.

5.2. Comparison with SPICE2

A comparison between the performance of SPICE3, Version 3C.1, and SPICE2, Version 2G.6, running on the same machine is described in this section. SPICE3 was compiled with the standard C compiler for Ultrix 3.0, while SPICE2 was compiled with the standard 4.2BSD "f77" FORTRAN compiler, both with optimization enabled. Comparisons with SPICE2 compiled using the advanced, DEC fort compiler are also included and a comparison with an industrial version of SPICE2 is included in Section 5.4.1.

SPICE3 was able to solve all circuits SPICE2 could solve, as well as a significant number of additional circuits. While the CPU time used by SPICE3 is not always strictly less than the time used by SPICE2, it is generally significantly less, and in the cases where it is greater, it is generally due to a change made to increase the number of circuits which can be solved or to increase the accuracy of the simulation.

5.2.1. Circuits which both SPICE2 and SPICE3 run

These circuits have all been run on both SPICE2 and SPICE3. To make the data more readable, first consider just the total CPU time used for the entire simulation, as shown in Figures 5.1(a) and (b).

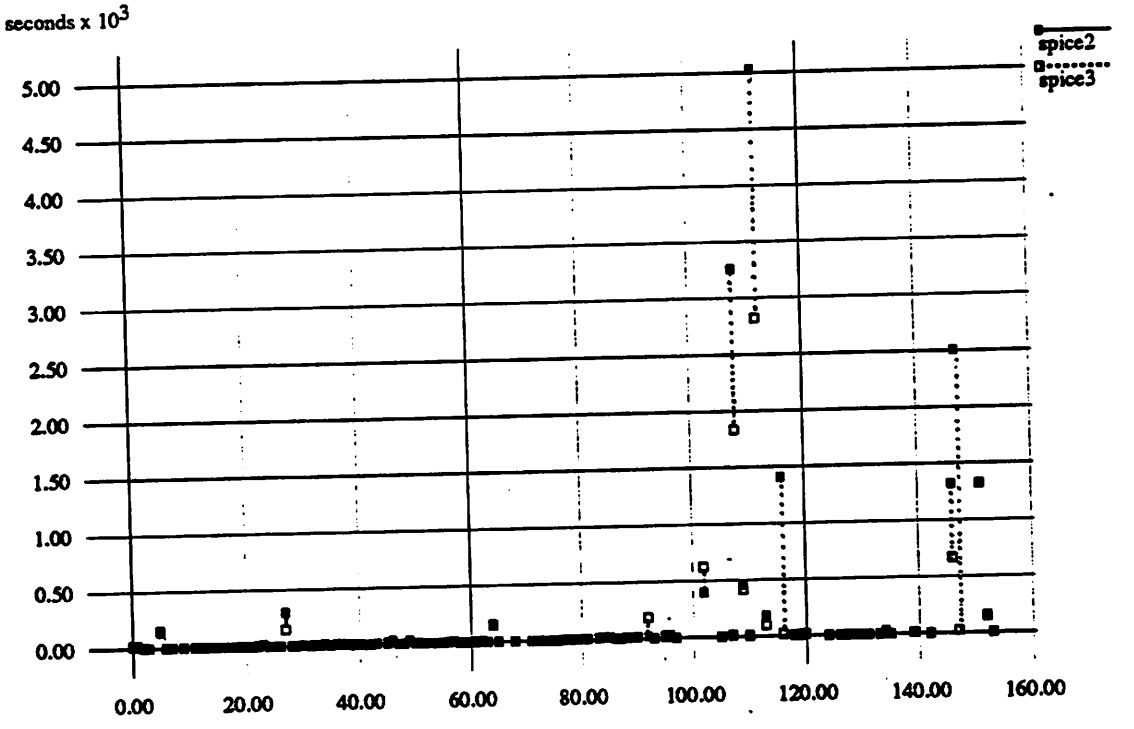

Figure 5.1(a)
Total CPU time for benchmark
circuits for SPICE2/f77 and SPICE3.

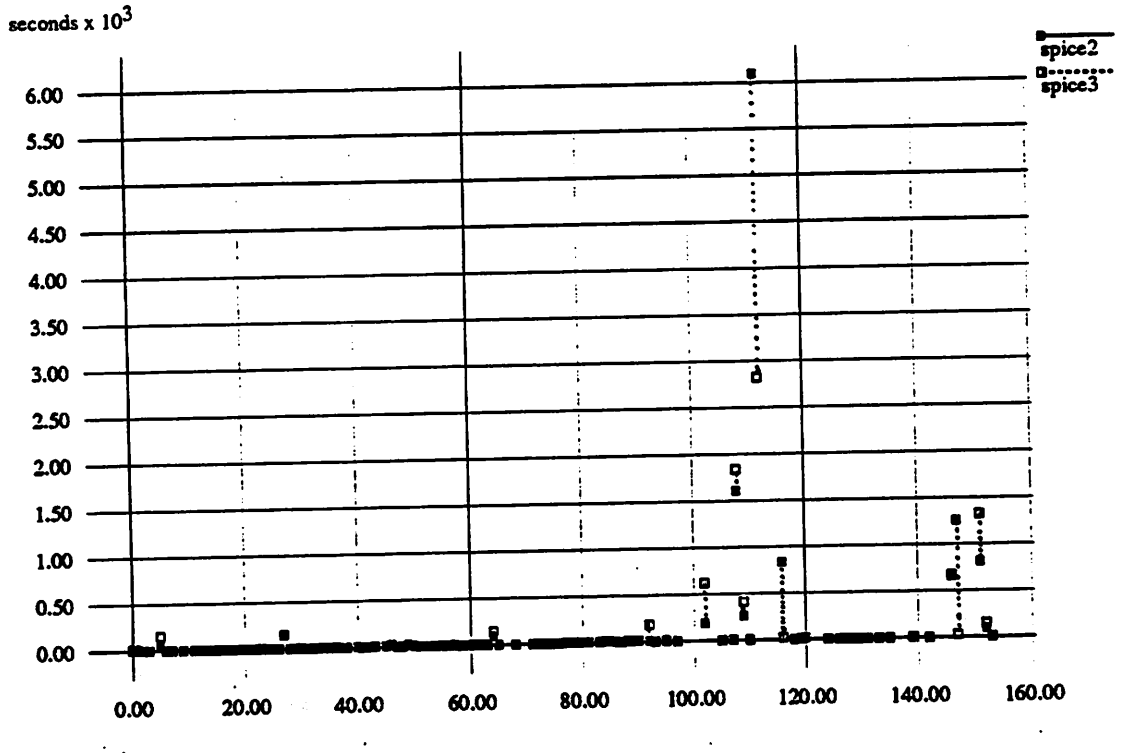

Figure 5.1(b)
Total CPU time for benchmark
circuits for SPICE2/fort and SPICE3.

SPICE3 averages a significantly lower CPU time for these circuits with very few examples where it takes longer, although its advantage is reduced when SPICE2 is compiled with the advanced FORTRAN compiler. There are four circuits with significantly higher CPU time for SPICE3 than for SPICE2: N9AT, C68DT, N27A3O, and N27A4O. In examining the results of runs of these circuits, it can be seen that SPICE2 had difficulty with the currents in circuit N9AT, resulting in severe ringing. SPICE3 does not exhibit this difficulty, producing smooth curves through the region, but at the expense of significantly more timepoints, both in this region and at later points in the analysis.

Circuits N27A3O and N27A4O are both variations on the same MOS amplifier. This particular MOS amplifier converges rapidly in SPICE2 due to an asymmetry in the limiting routine. This asymmetry can cause problems since it makes the convergence properties of a circuit depend on which

node the user designated the source and which the drain in an inherently symmetric device. SPICE3 has removed this asymmetry and as a result must use G_{min} stepping to locate the operating point in this exception case which results in over eight times as many iterations.

Circuit C68DT is a circuit that was exhibiting problems which were critically dependent on the size of its load capacitances. While the reason it takes significantly more timepoints, and thus more time, is not immediately obvious, it is probably related to the changes made in the capacitance model as described in Chapter 3. The SPICE3 analysis accepts approximately twice as many timepoints and rejects approximately four times as many as the SPICE2 analysis, but the resulting waveforms are almost indistinguishable.

Note that the results in Appendix G indicate that SPICE3 uses more time *per-iteration* than SPICE2 for the corresponding circuits, but that 285,000, or over 70% of the total iterations SPICE2 took for the entire set of simulations, are for a single circuit, Q2A1T, which, as a small circuit, has a very small per-iteration time of 4.93 seconds, thus skewing the per-iteration statistics. SPICE3 computes the solution to this same circuit in 4600 iterations with a somewhat smaller per-iteration time. If this single circuit is excluded from Figure A.13, the per-iteration CPU time for SPICE2 is increased to 138.66 ms under the f77 compiler and 101.55 ms under the fort compiler versus the 84.75 ms for SPICE3 compiled with the standard cc compiler.

Looking at the transient analysis results in more detail, consider the breakdown of simulation time into device evaluation and matrix solution as shown in Figures 5.2 and 5.3.

In all but a few cases, SPICE3 uses the same or less CPU time than SPICE2 for the same circuit. The only cases where the SPICE2 evaluation time is measurably less are bipolar circuits, where the device evaluation optimization techniques described for MOSFETs in Chapter 2 have not been applied and the evaluation code remains a straightforward translation of the corresponding SPICE2 FORTRAN code.

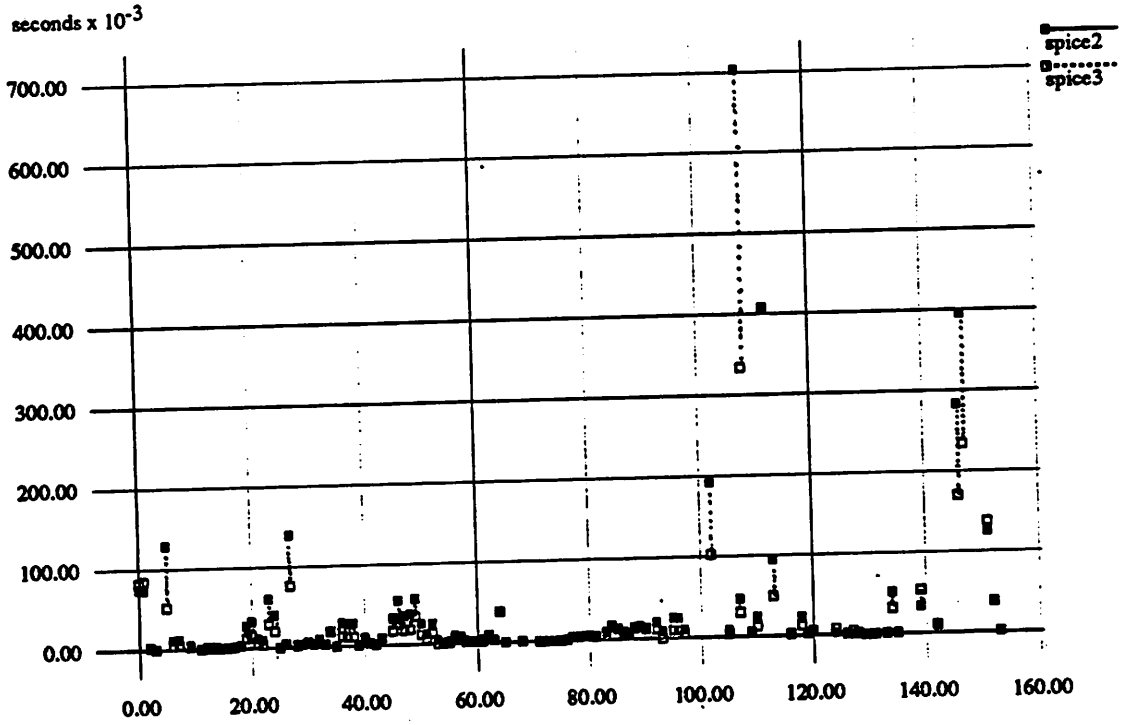

Figure 5.2(a)
Device evaluation time per iteration for SPICE2/f77 and SPICE3

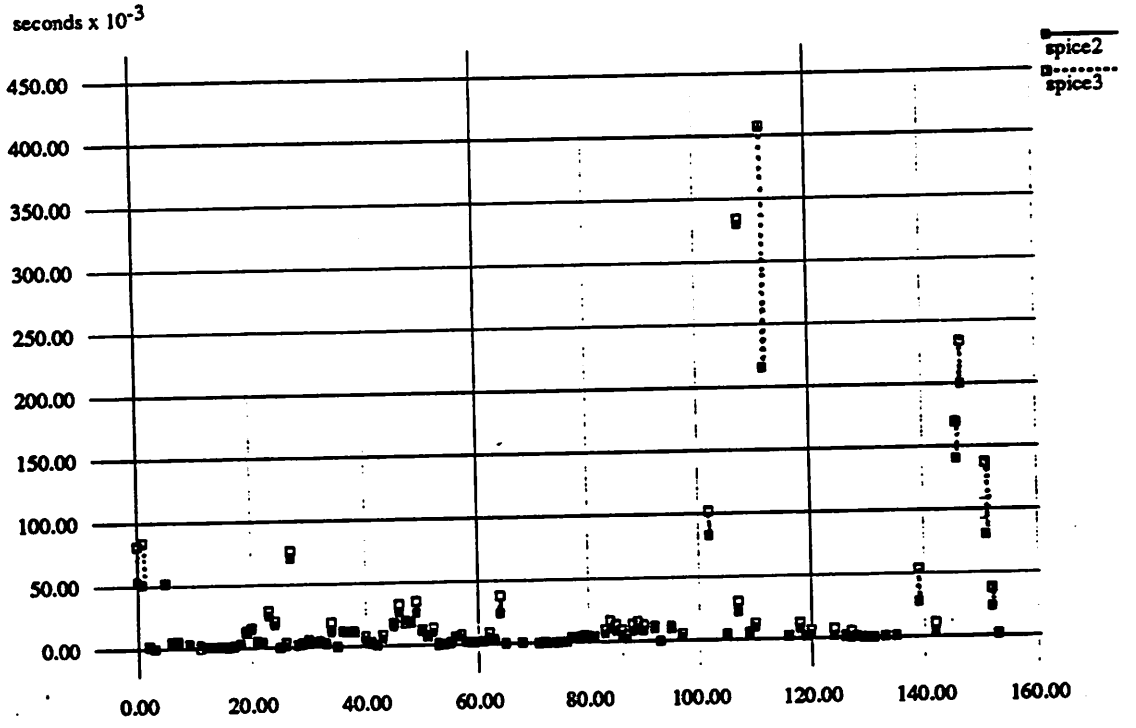

Figure 5.2(b)
Device evaluation time per iteration for SPICE2/fort and SPICE3

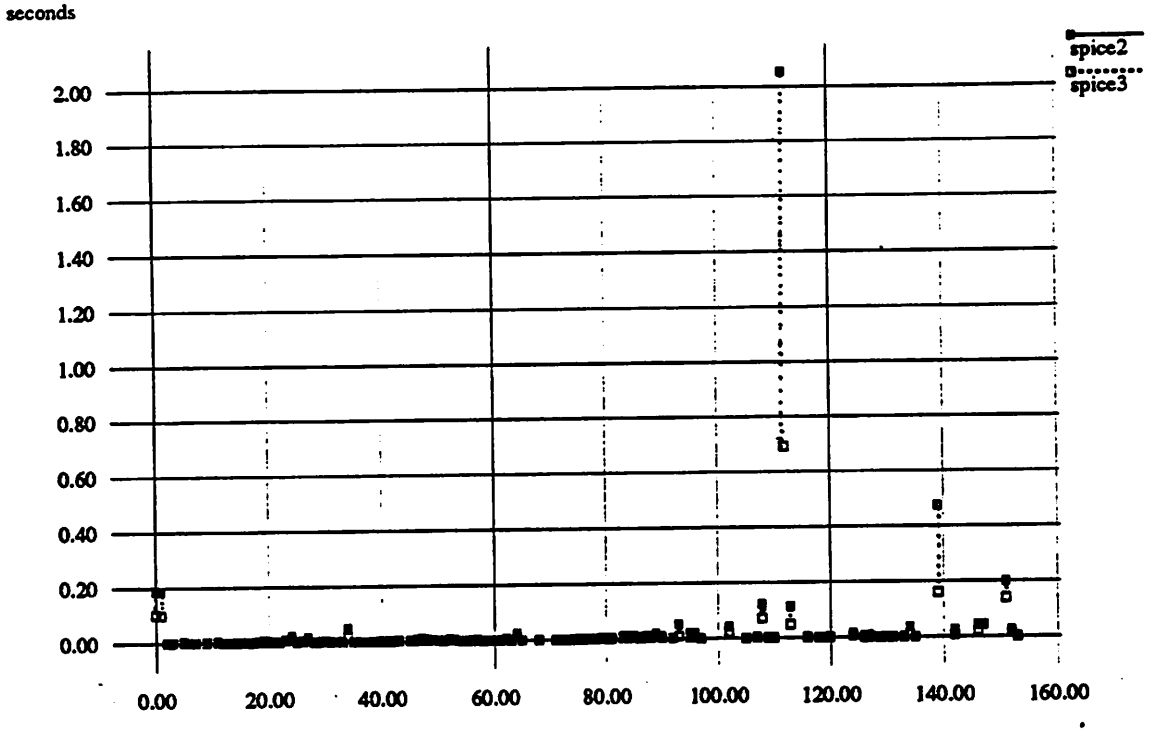

Figure 5.3(a)
Matrix solution time per iteration for SPICE2/f77 and SPICE3

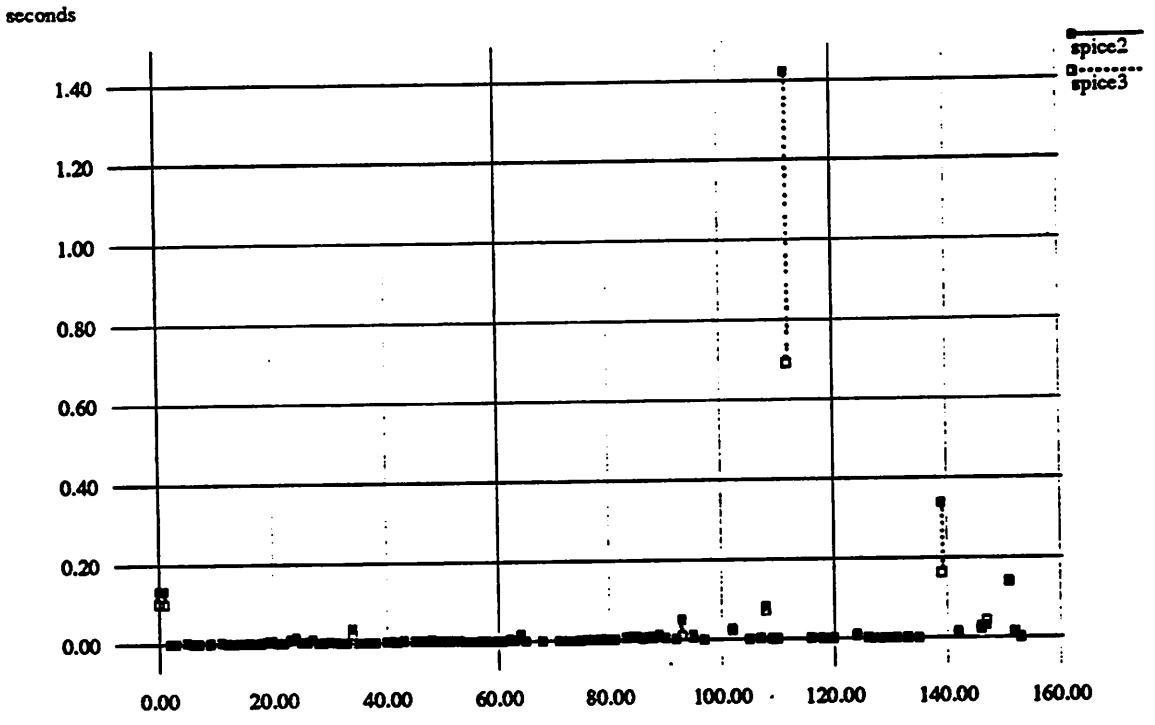

Figure 5.3(b)
Matrix solution time per iteration for SPICE2/fort and SPICE3

5.2.2. Circuits which only SPICE3 runs

A total of 27 circuits from the benchmark set converge in SPICE3 but not in SPICE2. Fourteen of these fail in SPICE2 because they use features that are not present in SPICE2. The features used include the controlled switches, GaAs MESFETs, MOSFETs using the BSIM model, named instead of numbered nodes, pole-zero analysis, and the ability to specify initial conditions for nodes internal to a subcircuit.

Of the remaining thirteen circuits, three of them, C38DA, C38D2A, and C6D1T, were all CMOS circuits for which SPICE2 could not find an operating point. The corrections to the Meyer model described in Chapter 3 allow SPICE3 to solve them. One circuit, O3A1T, failed in SPICE2 when the topology checker determined that the circuit was not properly formed, having only a single connection

to a node, but SPICE3 is able to solve the circuit, since the single connection to that node is a voltage source.

Four circuits, C18AT, C52AA, C31DT, and C1060MT all suffer from floating point errors during the dc solution. These four circuits all have problems in SPICE2 due to the asymmetric limiting used for MOSFETs. The asymmetric limiting does not properly limit all devices limiting is dependent upon the labeling of the drain and source nodes.

Two additional circuits, N3AT and C14D1T, caused SPICE2 to generate thousands of iterations during transient analysis and eventually stop because of iteration limits while SPICE3 found the transient solution with only a few hundred iterations. Both of these circuits have the characteristics necessary to generate difficulties with the Meyer model problem described in Chapter 3.

The last three circuits generate "timestep too small" errors in SPICE2, and again all have the characteristics which trigger the Meyer model errors described in Chapter 3.

5.2.3. Circuits which neither SPICE2 nor SPICE3 can run

Seven circuits still fail to converge in both SPICE2 and SPICE3 for a variety of reasons. Circuit C14DT is a BSIM circuit, thus SPICE2 can't run it, and it generates a floating point exception during a BSIM evaluation in SPICE3.

Circuit N804DT is a poorly formed circuit, having one node which is only connected to the gate of a MOSFET (a floating gate). SPICE2 detects this problem and aborts during the topological checking phase. SPICE3 performs the operating point analysis without difficulty since the dangling node has an initial condition specified for it, but then fails during the transient analysis when the conductance of that gate capacitor is insufficient to provide sufficient connectivity to ensure reasonable coefficients in the Jacobian, and the matrix package declares it nearly singular.

Circuit C82DT generates floating point overflows in SPICE2 while searching for the operating point. While SPICE3 does not encounter the same degree of difficulty, it is unable to find the operating point. Since this particular circuit contains a loop of fifteen inverters and no initial conditions or

"off" specifications are given, the operating point is extremely difficult to find under any conditions.

The circuit N1190MT encounters a floating point overflow problem in SPICE2 while looking for the operating point. SPICE3 properly finds the operating point, but runs into "timestep too small" errors during the transient analysis.

Circuit C119AT fails in SPICE2 due to a topology problem detected by the TOPCHK subroutine: a floating subcircuit exists which has no DC path to ground. SPICE3 is similarly unable to find a solution, although it reports a singular matrix and notifies the user of just one of the nodes involved.

Circuit C77MT causes both SPICE2 and SPICE3 to suffer a timestep too small error.

Finally, circuit H44AA is a pole-zero analysis which SPICE2 can not perform and for which SPICE3 eventually generates a convergence failure during one of the pole-zero searching iterations.

5.3. Comparison of Compilers

It is very difficult to isolate all of the factors that affect the performance of a program such as SPICE. One variable which can have a significant effect on the program's performance is the compiler which is used to generate the executable of the program. The machines primarily used for SPICE3 development were Digital Equipment Corporation VAX machines running a succession of releases of the UNIX operating system, both 4BSD and Ultrix.

SPICE2 was coded entirely in FORTRAN-66 and there were two real choices for the compiler; the BSD "f77" compiler and the DEC VAX FORTRAN/Ultrix "fort" compiler. The f77 compiler was a research tool from U.C. Berkeley, and the fort compiler is a highly-tuned commercial FORTRAN compiler with much-improved performance. The f77 compiler has improved in recent years but still provides significantly lower performance than fort. Thus some of the performance improvement in SPICE3 is due to the change from a poorly optimizing FORTRAN compiler to a somewhat better optimizing C compiler.

In the case of SPICE3, three different C compilers have been used, the standard BSD system C compiler (cc), the VAX C/Ultrix compiler (vcc), and the GNU[Stal88a] C compiler (gcc). These compilers

all provide somewhat different levels of optimization and performance as shown in the tables in Appendix G.

Unless clearly indicated otherwise, all of the comparisons in this dissertation have been based on the standard f77 and cc compilers since they are the most widely available and used on VAX/Unix systems.

Each compiler and language has different strengths and weaknesses. Fortran compilers have traditionally been highly optimizing and the language was designed with restrictions that make it relatively simple to make a number of optimizations. Unfortunately, these same restrictions make it difficult to write clean and readable versions of the complex code used by a program such as SPICE. C compilers, due to the flexibility of the language and, in particular, the use of pointers, have a much smaller set of optimizations that they can perform. Unfortunately, the language must assume that objects referenced through pointers may overlap, thus preventing many common subexpression optimizations and cacheing of values in registers. Thus, while C provides more efficiency by providing pointers for such operations as referencing elements of the sparse matrix, the reduced ability to perform common subexpression optimization reduces the performance advantage somewhat.

Using SPICE2 compiled with the f77 compiler as a reference, the fort compiler provides an average performance improvement of approximately 25 percent; SPICE3 compiled with the standard C compiler provides an average performance improvement of approximately 45 percent (measured on those circuits which both simulators can complete). Switching to more optimizing C compilers produces a further five to seven percent improvement. Finally, note that in addition to producing different performance results, the executable produced by different compilers produces different levels of success, with the executable compiled by fort unable to solve four circuits which the same source compiled with f77 solves, and the executable produced by vcc failing on three circuits for which cc and gcc-generated executables had no trouble.

industrial version of SPICE2 are reported in Figure 5.4(a). Many of the circuits run by SPICE3 could not be simulated by the industrial simulator, and so these circuits are not included in the figure.

Five circuits show a significantly better run time in the industrial simulator than in SPICE3. Two of these circuits are bipolar circuits where relatively little optimization has been performed to date in SPICE3. One is the C68DT circuit, described in Section 5.2.1, which exhibits load-capacitance-dependent problems. One is the N9AT circuit, also described in Section 5.2.1, which has an oscillation problem in SPICE2 which is not present in SPICE3. This circuit exhibits the same oscillation problem in the industrial simulator as it does in SPICE2. The industrial simulator appears to gain an edge over both SPICE2 and SPICE3 when the number of iterations required in the analysis becomes very large. Part of this advantage comes from the use of generated machine code for matrix

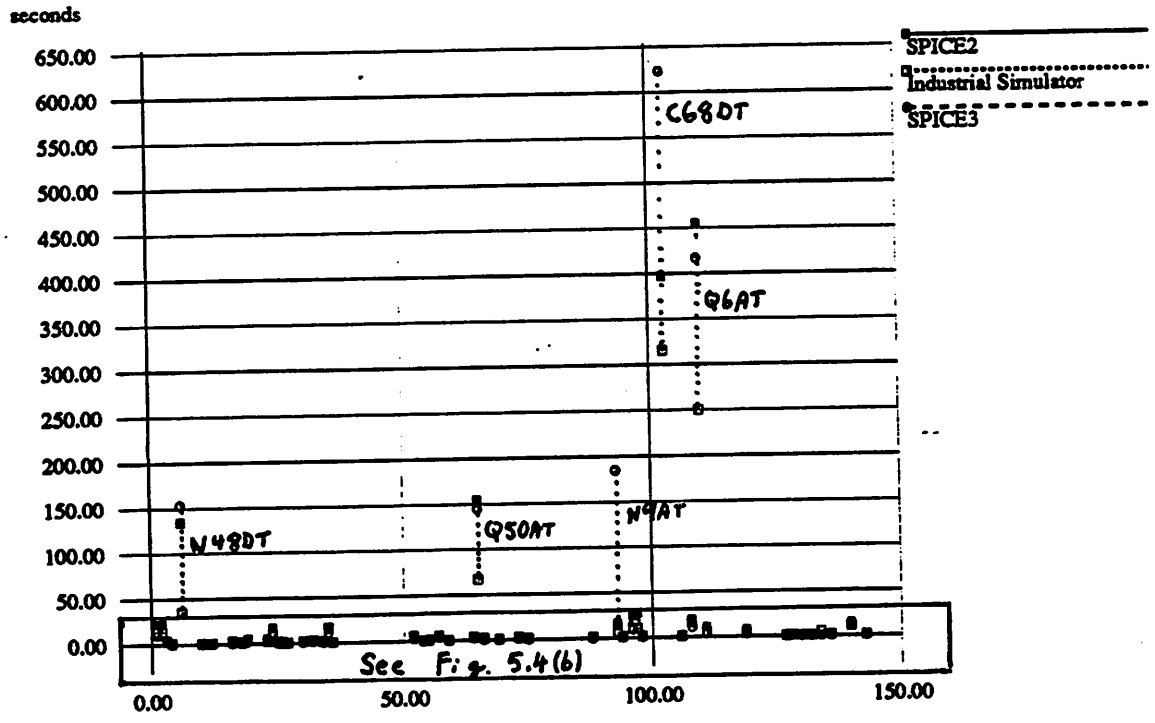

Figure 5.4(a)
Total CPU time comparison with industrial simulator

5.4. Comparison with Other Simulators

In order to get a better idea of how SPICE3 performs in comparison with other similar programs, it has been compared with two other simulators. The choice was made to test against both another general-purpose simulator and against a special-purpose simulator. The general-purpose simulator chosen is an industrial derivative of SPICE2 which has been revised for many years in a production environment and is highly tuned. The special-purpose simulator chosen is RELAX, Version 2.3b, a MOS simulator from U.C. Berkeley designed to simulate large digital circuits. Comparisons with SPLICE would also be possible, but in direct (non-relaxation) mode, SPLICE and RELAX use similar algorithms and data structures.

5.4.1. Comparison with an Industrial Circuit Simulator

Comparing an industrial circuit simulator with SPICE3 presents a number of problems. The simulator chosen has a slightly different input format than SPICE3 or SPICE2, so circuits must be translated from one format to another. In addition, the simulator results were only available for a VAX 8800, while the SPICE3 results were all from a VAX 8650. Hence, there is a slight difference in performance between the two simply because of the difference between the machines. To compensate for this difference, the benchmark set was run on a VAX 8800 under the VAX FORTRAN/ULTRIX compiler and compared with the results from running the same set on the VAX 8650 with the same compiler. This comparison, averaged over more than 100 runs for 10,000 seconds of CPU time, yielded a performance ratio between these two machines of approximately 1.005; thus a run of SPICE on the VAX 8650 could be expected to take approximately 1.005 times as long as it does on the VAX 8800.

After a fairly simple initial circuit translation, 90 circuits from the SPICE3 benchmark set could be run without input errors. Unfortunately, because of differences in the accounting information available from the simulators, 39 of these circuits do not produce output statistics which can be compared with those produced by SPICE3. After correcting the VAX 8800 simulation time by the proper factor to produce estimated VAX 8650 times, the total CPU times for SPICE2, SPICE3, and the

solution[Cohe76a] in the industrial program. This technique from SPICE2 has not been implemented in SPICE3, thought it could be, and it provides a significant reduction in matrix decomposition time for long transient runs.

When the region at the bottom of Figure 5.4(a) is magnified as in Figure 5.4(b), it is evident that SPICE3 has a comparable or smaller run time for all the examples in this region. Also note that the six largest run time examples for SPICE2 and SPICE3 could not be run with the industrial simulator, most of them failing with "timestep too small" or "no convergence in dc analysis" errors; thus results for these examples are not shown in this figure. On the 51 circuits that both programs can solve, the industrial simulator shows an average of a 50% run-time improvement over SPICE3 due to the 5 large circuits, but if these are excluded, the SPICE3 showed an average of 35% run-time

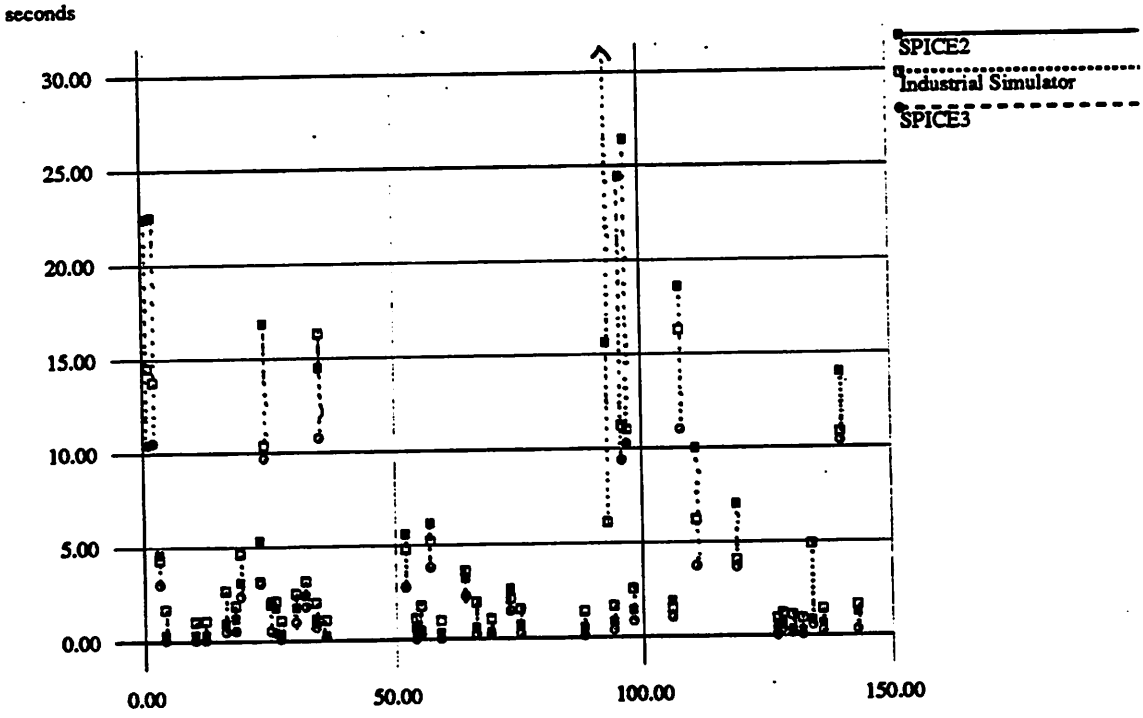

Figure 5.4(b)
Total CPU time comparison with industrial simulator

improvement over this industrial version of SPICE2.

5.4.2. Comparison with RELAX

RELAX is a simulator written at U. C. Berkeley which uses both direct methods and a modified waveform relaxation algorithm[Whit85a, Newt83a]. RELAX was designed primarily as a testbed for the development of waveform relaxation algorithms for the fast transient simulation of MOS integrated circuits but also incorporates direct methods for dc solutions and for its subcircuit evaluation. For this comparison, RELAX has been used in the mode where it uses direct methods for the entire circuit. While RELAX is a useful simulator for many circuits, it does have limitations that SPICE3 does not. Specifically, RELAX does not have models for many of the devices in SPICE3, including bipolar junction transistors, non-grounded voltage sources, and transmission lines. RELAX is also designed only for transient solution with operating point if required, not for any of the other types of analysis performed by SPICE3, such as ac, pole-zero, or dc transfer curve. It addition, RELAX requires that each node of a circuit include a capacitor to ground and inserts one if it is absent. While these limits still permit it to be used to analyze a large number of circuits, it is a significantly smaller set than SPICE3 was designed to solve.

An additional difference between RELAX and SPICE3 is the choice of the default parameters which control the simulation. SPICE3 is configured to work well on most circuits at the expense of a longer solution on simple circuits, while RELAX is configured with default parameters which are more suitable for digital circuits using, for example, a 1% relative error limit in the truncation error calculations instead of the more conservative 0.1% used in SPICE3. However, the trtol parameter in SPICE3 makes its 0.1% comparable to a 0.7% value for RELAX. Note further that RELAX uses quite different criteria for its convergence tests and instead of performing the test:

$$\text{DELTAV} \leq \text{reltol} \times \text{MAX}(V_{old}, V_{new}) + \text{abstol}$$

for each node voltage individually, it instead performs the test:

$$\text{DELTAV} \leq \text{reltol} \times \text{MAX}(\text{allV's}) + \text{abstol}$$

This means that in a circuit with 5 volt power supplies and a relative tolerance of 0.001 (the default

in RELAX) voltages are considered to converge if they change by less than 0.005 volts regardless of the signal levels present on that node, allowing RELAX to reach this looser convergence criteria in fewer iterations. This also makes the abstol parameter nearly useless, since it will normally be much smaller than reltol times one of the voltage rails. Similarly, the truncation error calculation is performed only once, using the maximum voltage computed above rather than once for each individual capacitor with the tolerances computed for that capacitor based on the charge and current through it. These differences will affect both the number of iterations used to obtain convergence and the size of the timestep used.

Due to the more general-purpose design of SPICE3 which accommodates all of these additional features, its performance cannot be expected to match RELAX on circuits for which RELAX was designed to operate most effectively, although the differences should not be too significant when the programs are forced to work with comparable limits.

If the limits on SPICE3 are relaxed, SPICE3 can also produce results in significantly less time. As an example, consider the results of varying the convergence criteria and local truncation error limits while running the N698DT circuit as shown in Figure 5.5.

Examining the outputs from these runs, significant deviations from the results with the tightest tolerances do not occur until reltol is increased into the 10% to 50% range, thus SPICE3 can obtain the desired results *for this circuit* using a reltol 50 times that used by default with a resulting 30% decrease in iteration count. Again, while reltol could be relaxed for a class of circuits, gaining significant performance advantages, it would result in a less robust program over its full range of uses.

5.4.2.1. RELAX comparison circuits

A small set of circuits was chosen to convert to RELAX format and run in both SPICE3 and RELAX for comparison. The most recent released version of RELAX, RELAX 2.3b, running on the same VAX 8650 with Ultrix 3.0 was used for all comparisons. These circuits were selected to provide a range of circuit sizes, from large to small, while remaining within the RELAX circuit constraints. The circuits chosen were N698DT, a large digital filter, C205AT, a phase lock loop, C640DT, a serial

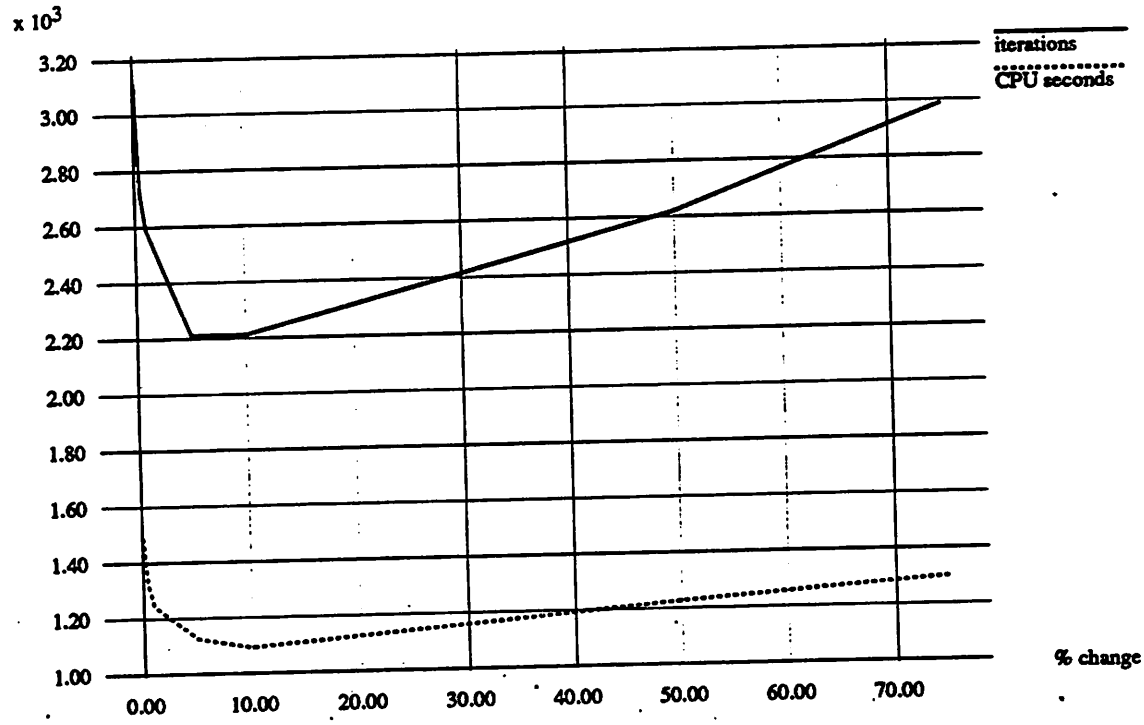

Figure 5.5
Results of varying tolerances on Circuit N698DT
(Run on a VAX 8800 running Ultrix 3.0)

register, C1060MT, a large CMOS memory circuit, and N12AT, a small NMOS memory cell. Conversion of these circuits requires a number of decisions as to how to map parameters. RELAX has a number of different parameters for determining convergence and acceptable error levels, while SPICE3 has a smaller set of parameters for the same purposes. RELAX uses *nrvabs* as the absolute upper limit on the voltage difference allowable between iterations at a node without forcing an additional iteration. *nrcabs* is the corresponding limit on current changes between iterations. There are also separate relative limits *nrvrel* and *nrcrel* for the voltage and current changes. Finally, there are absolute and relative node voltage errors acceptable to the local truncation error control algorithm that controls step size: *lteabs* and *lterel*. In SPICE3, there is a single parameter, *reltol*, which determines all of the relative error tolerances and a single absolute tolerance for each unit, *vntol* for voltages, *abstol* for

current, and *chgtol* for charge. These tolerances are used for both convergence testing *and* as maximum acceptable error estimates for the local truncation error control.

In converting the input decks from SPICE3 to RELAX, choices must be made as to what to change these parameters to, since both simulators have defaults and in many of the circuits tested these defaults were changed explicitly as well.

The overall results are presented in Table 5.6, but some explanation of these numbers is in order. Each of the circuits was run twice for each simulator, once using that simulator's default tolerances and once using the tolerances specified by the circuit designer. In each case, the only tolerances adjusted in RELAX were the Newton-Raphson convergence tolerances and not the truncation error stepsize control parameters, since RELAX strongly recommends that these parameters not be tied directly to one another due to the algorithms used for truncation error calculation. The RELAX

Table 5.6
Comparison of SPICE3 and RELAX performance on the selected set of circuits.

Circuit	Simulator	Tolerances	Total iterations	Total CPU	Time per iteration
N698DT	RELAX	default	4901	485.20	0.099
N698DT	RELAX	specified	4917	470.62	0.096
N698DT	SPICE3	default	5627	2846.05	0.506
N698DT	SPICE3	specified	3118	1491.62	0.478
C205AT	RELAX	default†	no convergence‡	no convergence	
C205AT	SPICE3	default	6940	993.12	0.143
C640DT	RELAX	default	1258	1031.59	0.820
C640DT	RELAX	specified	1311	1060.84	0.809
C640DT	SPICE3	default	1184	1250.50	1.056
C640DT	SPICE3	specified	1611	1839.28	1.142
C1060MT	RELAX	default	1510	996.89	0.660
C1060MT	RELAX	specified	no convergence	no convergence	
C1060MT	SPICE3	default	1859	2051.94	1.104
C1060MT	SPICE3	specified	788	735.25	0.933
N12AT	RELAX	default	457	1.22	0.002
N12AT	RELAX	specified	514	1.77	0.003
N12AT	SPICE3	default	258	2.87	0.011
N12AT	SPICE3	specified	240	2.66	0.011

† No tolerances specified
‡ converges in a newer, unreleased version of RELAX[Webb89a].

documentation recommends that the truncation error parameters be at least an order of magnitude larger than the convergence criteria. Also note that RELAX uses Jacobian evaluation bypass, computing and inverting the Jacobian only every third iteration, while SPICE3 uses inactive device bypass to skip the repeated evaluation of inactive devices.

Looking at these results, it is immediately apparent that there are significant performance differences between SPICE3 and RELAX. A detailed investigation of the differences between the two was conducted and showed a number of interesting differences. An overall profile of the two simulators while running circuits N698DT and C640DT shows that approximately 100 seconds of additional time is spent in the matrix package for reordering the matrix. Clearly the more sophisticated sparse matrix package "sparse^{Kund88a}" used in RELAX is able to save about 20% of the time spent by the SMP matrix package in SPICE3.

Other areas where RELAX outperforms SPICE3 in this comparison are the SPICE3 inactive device bypass and limiting routines, which consume approximately 13% and 6% respectively of SPICE3's simulation time. The Jacobian bypass in RELAX requires no significant computation time and the node-based limiting in RELAX only takes about 10 seconds, although they may both increase the iteration count. The integration scheme used in SPICE3 requires the integration of five capacitors for each of the 698 MOSFETs, plus the 385 constant capacitors, while RELAX only needs to integrate the node capacitance, another saving of approximately 200 seconds.

An additional part of the difference between SPICE3 and RELAX in these circuits can also be attributed to the bypass of components within a device. RELAX detects when the oxide capacitance is zero and completely skips all of the oxide capacitance related code. SPICE3 does not yet check for this special case, but instead performs all of the appropriate calculations and multiplies the result by zero. Table 5.7 shows the results of considering just this one effect on circuit N698DT.

Table 5.7
Effect of Oxide Capacitance Bypass on circuit N698DT

Program	Tox bypass?	Tolerances	Tran. Iterations	Tran CPU	CPU/iter(ms)
RELAX	Yes	Given	4698	580.12	123.5
RELAX	No	Given	4698	813.22	173.1
RELAX	Yes	Spice defaults	4698	574.91	122.2
RELAX	No	Spice defaults	4698	822.04	175.0
SPICE3	Yes	Given	3417	1365.16	399.5
SPICE3	No	Given	3417	1986.78	581.1
SPICE3	Yes	Spice defaults	5592	2180.77	390.0
SPICE3	No	Spice defaults	5592	3251.67	581.5

This figure was generated using specially modified test versions of RELAX and SPICE3 to explicitly skip or perform the bypass operation. It is clear from these numbers that for circuits with no oxide capacitance, both SPICE and RELAX can save approximately 30% of their per-iteration device evaluation time by making this optimization, but only RELAX currently takes advantage of it. Clearly this is an optimization that must be integrated into SPICE3 in the near future.

The additional code required by SPICE3 every iteration for the computation of parameters which are not pre-computed and stored, such as the per-device temperature-dependent V_t needed by the temperature variation code in SPICE3, adds about 7% to the SPICE3 time. Approximately 30% of the difference appears to be coming from a difference in coding style between SPICE3 and RELAX. RELAX was written as a test for relaxation simulation, while SPICE3 was written as a general testbed for circuit simulation, and more emphasis was placed on code readability and clarity of all algorithms, including the device models. As a result, the corresponding code in the two programs is frequently simpler to understand in SPICE3, but executes more efficiently in RELAX. For comparison, consider the following code fragments from RELAX and SPICE.

```
if(vbs <= 0) {
        here->MOS1gbs = SourceSatCur / vt ;
        here->MOS1cbs = here->MOS1gbs * vbs ;
        here->MOS1gbs += ckt->CKTgmin;
} else {
        evbs = exp(MIN(MAX_EXP_ARG , vbs/vt)) ;
        here->MOS1gbs = SourceSatCur * evbs / vt + ckt->CKTgmin ;
        here->MOS1cbs = SourceSatCur * (evbs - 1) ;
}
```

Figure 5.8
SPICE3 code fragment

```
if(fptr->isas == 0.0) { gbs = 0.0; ibs = 0.0; }
else {
  if(vbs >= fptr->svmax) {
    gbs = dptr->maxcond;
    ibs = dptr->imax - fptr->isas + gbs * (vbs - fptr->svmax);
  }
  else if(vbs > 0.0) {
    gbs = fptr->isas * exp(vbs * dptr->ovt);
    ibs = gbs - fptr->isas;
    gbs *= dptr->ovt;
  }
  else { gbs = fptr->isas * dptr->ovt; ibs = gbs * vbs; }
};
```

Figure 5.9
RELAX code fragment

In the most common case, SPICE3 will perform one exponential, three divisions (two with a very good compiler), two multiplications, one addition and two subtractions. RELAX for the same case will perform one exponential, three multiplications, and one subtraction, thus replacing one of the divisions with a multiplication, saving two multiplications, one addition, and one subtraction. These savings are achieved by additional pre-computation of device parameters and such optimizations as computing and saving $\frac{1}{V_t}$ and multiplying by that instead of dividing by V_t during the analysis. These same types of optimizations are possible in SPICE3, but make the algorithms somewhat less

clear, and until the SPICE models are fully documented and carefully re-implemented rather than simply translated from one language to another with bug fixes applied, this level of optimization is not appropriate.

Note that in the comparison circuits, circuit C1060MT came from an industrial source with an absolute voltage tolerance of 10μv and a relative tolerance of 0.004, or 0.4%. Using these parameters, along with the absolute current tolerance of 50picoamps, RELAX is unable to find a DC solution. Tightening these tolerances to the defaults for RELAX, the circuit can be solved, although at the cost of a higher simulation time. Similarly, circuit C205AT came with no tolerances specified by the user and could not be solved by RELAX.

CHAPTER 6

Conclusions

In this dissertation, a number of algorithms and techniques have been presented which can be used to improve significantly the convergence properties and run time performance of a direct-method circuit simulator, such as SPICE. These algorithms have been implemented and tested in a new circuit simulation framework which has been designed as a flexible testbed for both circuit simulation algorithm and model development. A summary of the major results and possible directions for additional work are presented below.

Initially, the SPICE3 framework was profiled and a detailed analysis of the areas where the program spent significant time was performed. Based on this profiling information, effort was concentrated on areas of the simulation where performance improvements could have the greatest effect on the overall simulation time. Changes to the device modeling routines and to some of the overall simulation algorithms resulted in significant speed improvements and much improved convergence properties.

While the speedups obtained to date have been significant, additional improvements can be expected if additional effort is applied to the optimization of the device models. In this project, little work was performed on the optimization of the bipolar models, so work in that direction should provide enhancements fairly quickly. Additional performance improvements in such areas as the MOS models are certainly possible if all special cases are considered, such as a bypass of the gate capacitance calculations for circuits like the N698DT example, where the gate capacitance is provided as an external capacitance instead of the nonlinear gate capacitance provided as part of the MOS model. As was observed in the comparison with RELAX, this could result in a reduction in the MOS evaluation time by as much as 10% for Level 1 MOSFETs with no gate capacitance.

A set of difficult examples was used to illustrate a variety of convergence problems and the cause of those problems was analyzed. As a result of this analysis, additional algorithms were added to SPICE3, along with corrections to a number of device models so as to allow most of these difficult circuits to converge. Some circuits still fail to converge, indicating further work is still needed if all of the causes of convergence failures or "timestep too small" errors are to be found.

All of the above research and resulting changes performed in this project required a flexible software framework and an overview of the SPICE3 program architecture, designed to facilitate this experimentation with algorithms, device models, and interfaces, was described. This architecture allows SPICE3 to be used as a subroutine library within a larger system and also allows the user interface from SPICE3 to be used for other simulators, thus increasing the compatibility between simulators and reducing the need to "re-invent" the necessary user interfaces every time. Details of this architecture and the steps needed to add to the program are included as appendices.

A study was carried out to compare the performance of SPICE3 with both a comparable general-purpose circuit simulator, SPICE2, and against a special-purpose simulator, RELAX. RELAX is a MOS only simulator that can use both direct methods and waveform relaxation techniques. Although RELAX can simulate many MOS digital circuits in somewhat less time, SPICE3 demonstrated greater reliability over a wider range of circuits, with a significant performance improvement over SPICE2.

The result of this work is both a faster, more reliable, and more flexible circuit simulator that can also serve as a framework for the design and implementation of new analyses and new device models, as well as an understanding of what factors lead to run-time and convergence issues for direct-method circuit simulation.

APPENDIX A

The Front End to Simulator Interface

To reduce this memo to a manageable size, this appendix has been placed in a separate memo. This memo, *The Front End to Simulator Interface*, is available as UCB/ERL Memorandum M89/43.

APPENDIX B

Data Structures

To reduce this memo to a manageable size, this appendix and Appendix C have been placed in a separate memo. This memo, *The SPICE3 Implementation Guide*, is available as UCB/ERL Memorandum M89/44.

APPENDIX C

Packages

To reduce this memo to a manageable size, this appendix and Appendix B have been placed in a separate memo. This memo, *The SPICE3 Implementation Guide*, is available as UCB/ERL Memorandum M89/44.

APPENDIX D

Adding a Device

To reduce this memo to a manageable size, this appendix and Appendix E have been placed in a separate memo. This memo, *Adding Devices to SPICE3*, is available as UCB/ERL Memorandum M89/45.

APPENDIX E

The Device to Simulator Interface

To reduce this memo to a manageable size, this appendix and Appendix D have been placed in a separate memo. This memo, *Adding Devices to SPICE3*, is available as UCB/ERL Memorandum M89/45.

APPENDIX F

SPICE2 Compatible Input Language

To reduce this memo to a manageable size, this appendix has been placed in a separate memo. This memo, *SPICE3 Version 3C1 Users Guide*, is available as UCB/ERL Memorandum M89/46.

APPENDIX G

Benchmark Circuits

To reduce this memo to a manageable size, this appendix has been placed in a separate memo. This memo, *Benchmark Circuits Results for SPICE3*, is available as UCB/ERL Memorandum M89/47.

References

Acto70a.

Acton, Forman S., *Numerical Methods that Work*, Harper and Row, New York (1970).

Anto68a.

Antosiewicz, H. A., "Newton's Method and Boundary Value Problems," *Journal of Computer Systems Science* 2(2) pp. 177-202 (1968).

Berk84a.

Berkeley, Computer Science Division, Department of Electrical Engineering and Computer Science, U.C., "prof - display profile data," in *Unix User's Manual*, U.C. Berkeley, Berkeley, Ca. (March, 1984).

Berk84b.

Berkeley, Computer Science Division, Department of Electrical Engineering and Computer Science, U.C., "gprof - display call graph profile data," in *Unix User's Manual*, U.C. Berkeley, Berkeley, Ca. (March, 1984).

Bill83a.

Billingsley, Giles, "KIC," MS Report, Department of Electrical Engineering and Computer Sciences of Universoty of California at Berkeley (Fall, 1983).

Bray72a.

Brayton, Robert K., Gustavson, Fred G., and Hachtel, Gary D., "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Backward Differentiation Formulas," *Proceeding of the IEEE* 60 pp. 98-108 (1972).

Chri87a.

Christopher, Wayne A., *Nutmeg Programmer's Guide*, U.C. Berkeley, Berkeley, Ca. (April, 1987).

Chua75a.

Chua, Leon O. and Lin, Pen-Min, *Computer-aided Analysis of Electronic Circuits: Algorithms & Computational Techniques*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1975).

Cohe76a.

Cohen, E., "Program Reference for SPICE2," *Electronics Res. Lab.*, University of California, Berkeley, (June 1976).

Deva85a.

Devadas, Srinivas, *CSIM User's Manual and Report*, University of California, Berkeley (June, 1985).

Gear68a.

Gear, C. W., *The Control of Parameters in the Automatic Integration of Ordinary Differential Equations*, Department of Computer Science, University of Illinois (1968).

Gear69a.

Gear, C. W., "The Automatic Integration of Stiff Ordinary Differential Equations," *Information Processing*, pp. 187-193 (1969).

Gett86a.

Gettys, Jim, Newman, Ron, and Fera, Tony Della, *Xlib - C Language X Interface, Protocol Version 10*, Massachusetts Institure of Technology, Cambridge, Massachusetts (November 16, 1986).

Grah82a.

Graham, A. L., Kessler, P. B., and McKusick, M. K., "gprof: A Call Graph Execution Profiler," *Proceeding of the SIGPLAN 1982 Symposium on Compiler Construction, SIGPLAN Notices* 17(6) pp. 120-126 (June, 1982).

Ho75a.

Ho, C. W., Ruehli, A. E., and Brennan, P. A., "The Modified Nodal Approach to Network Analysis," *IEEE Transactions on Circuits and Systems* CAS-22 pp. 504-509 (June 1975).

Kele88a.

Kelessoglou, Theologos M. and Pederson, Donald O., "A Knowledge-Based SPICE Environment for Improved Convergence and User Friendliness," *Proceedings IEEE Custom Integrated Circuits Conference*, pp. 3.1.1-3.1.4 (May, 1988).

Kern78a.

Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1978).

Kund85a.

Kundert, Kenneth S., "Sparse Matrix Techniques and their Application to Circuit Simulation," in *Circuit Analysis, Simulation and Design*, ed. Albert E. Ruehli,North-Holland Publishing Co. (1985).

Kund88a.

Kundert, Kenneth S. and Sangiovanni-Vincentelli, Alberto, *Sparsel.3 : A Sparse Linear Equation Solver*1988.

McCa88a.

McCalla, William J., *Fundamentals of Computer-Aided Circuit Simulation*, Kluwer Academic Publishers, Boston (1988).

Meye71a.

Meyer, J. E., "MOS Models and Circuit Simulation," *RCA Review* **Volume 32**(March, 1971).

Nage75a.

Nagel, L., "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *ERL Memo* **UCB/ERL M75/520**University of California, Berkeley, (May 1975).

Nage73a.

Nagel, L. W. and Pederson, D. O., "Simulation Program with Integrated Circuit Emphasis (SPICE)," *Proceedings 16th Midwest Symposium on Circuit Theory*, (April 12, 1973).

Nara87a.

Narayanaswamy, Shankar, "SPICE3 Pulse-Voltage-Source modeling," EECS 199 Project Report, U.C. Berkeley, Berkeley, Ca. (January 1987).

Newt83a.

Newton, A. Richard and Sangiovanni-Vincentelli, Alberto, "Relaxation-Based Electrical Simulation," *IEEE Transactions on Electron Devices* **ED-30**(9) pp. 1184-1206 (September, 1983).

Newt77a.

Newton, A. R. and Pederson, D. O., "Analysis Time, Accuracy and Memory Requirement Tradeoffs in SPICE2," *Proceedings of the Eleventh Annual Asilomar Conference on Circuits, Systems and Computers*, (November, 1977).

Newt76a.

Newton, A. R. and Taylor, G. L., "BIASL.25 - An MOS Circuit Simulation Program for a Programmable Calculator," *Proceedings 10th Annual Conference on Circuits, Systems, and Computers*, pp. 280-283 (November, 1976).

Newt81a.

Newton, A. R., Pederson, D. O., Sangiovanni-Vincentelli, A. L., and Sequin, C. H., "Design Aids for VLSI: The Berkeley Perspective," *IEEE Transactions on Circuits and Systems* **CAS-28**(7) pp. 666-680 (July, 1981).

Orte70a.

Ortega, James M. and Rheinboldt, Werner C., *Iterative solution of nonlinear equations in several variables*, Academic Press, New York (1970).

Pres86a.

Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T., *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York (1986).

Quar84a.

Quarles, Thomas and Gyurcsik, Ronald, "Predictor-Corrector Based LTE Calculations in

SPICE," EECS 221 Project Report, U.C. Berkeley, Berkeley, Ca. (February 1984).

Quar83a.

Quarles, T., "The SPICE 3 Circuit Simulator," Masters report, University of California, Berkeley, ERL Memo ERL-M592 (December 1983).

Ritc74a.

Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," *Communications of the ACM* 17(7) pp. 365-375 (July, 1974).

Saka87a.

Sakallah, Karem A., Yen, Yao-tsung, and Greenberg, Steve S., "The Meyer Model Revisited: Explaining and Correcting the Charge Non-Conservation Problem," *Proceedings IEEE International Conference on Computer-Aided Design*, pp. 204-207 (November, 1987).

Sale87a.

Saleh, Resve, "Nonlinear Relaxation Algorithms for Circuit Simulation," *Electronics Res. Lab.*, University of California, Berkeley, (April 1987).

Sale86a.

Saleh, Resve, *Private communication*1986.

Sche88a.

Scheifler, Robert W., Gettys, Jim, and Newman, Ron, *X Window System C Library and Protocol Reference*, Digital Press (1988).

Sheu84a.

Sheu, Bing J., Scharfetter, Don L., Hu, Chenming, and Pederson, Donald O., *A Compact IGFET Charge Model*, Electronics Research Laboratory, U.C. Berkeley, Berkeley, Ca. (February, 1984).

Stal88a.

Stallman, Richard M., *Using and Porting GNU CC*, Free Software Foundation, Inc. (December, 1988).

Stat87a.

Statz, Hermann, Newman, Paul, Smith, Irl W., Pucel, Robert A., and Haus, Hermann A., "GaAs FET Device and Circuit Simulation in SPICE," *IEEE Transactions on Electron Devices* **ED-34, Number 2** pp. 160-169 (February, 1987).

Vlad80a.

Vladimirescu, A. and Liu, S., "The Simulation of MOS Integrated Circuits Using SPICE2," *ERL Memo. No. UCB/ERL M80/7*, (Feb. 1980).

Ward78a.

Ward, D. E. and Dutton, R. W., "A Charge-Oriented Model for MOS Transistor Capacitances," *IEEE Journal of Solid-State Circuits* **SC-13**(No. 5)(October, 1978).

Webb86a.

Webber, Don, *Private communication*1986.

Webb89a.

Webber, Don, *Private communication*Feb. 11, 1989.

Week73a.

Weeks, W. T., Jimenez, A. J., Mahoney, G. W., Mehta, D., Qassemzadeh, H., and Scott, T. R., "Algorithms for ASTAP -- A Network Analysis Program," *IEEE Transactions on Circuit Theory* **CT-20** pp. 628-634 (November, 1973).

Whit85a.

White, Jacob, "The Multirate Integration Properties of Waveform Relaxation, with applications to Circuit Simulation and Parallel Computation," *Electronics Res. Lab.*, University of California, Berkeley, (November 1985).

Yang82a.

Yang, Ping and Chatterjee, Pallab K., "Spice Modeling for Small Geometry MOSFET Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **CAD-1**(No. 4) pp. 169-182 (October, 1982).

Yang83a.

Yang, Ping, Epler, Berton D., and Chatterjee, Pallab K., "An Investigation of the Charge Conservation Problem for MOSFET Circuit Simulation," *IEEE Journal of Solid-State Circuits* Vol SC-18(No. 1) pp. 128-138 (February, 1983).

Yang80a.

Yang, Ping, *An Investigation of Ordering, Tearing, and Latency Algorithms for the Time-Domain Simulation of Large Circuits,* Department of Electrical Engineering, University of Illinois at Urbana-Champaign (August 1980).