A DIAL-AN-OPERATOR APPROACH TO SIMULATION OF IMPURITY DIFFUSION IN SEMICONDUCTORS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Daniel W. Yergeau
October 1999

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Robert W. Dutton
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Kincho Law
(Civil Engineering)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Peter Pinsky
(Mechanical Engineering)

Approved for the University Committee on Graduate Studies:

Abstract

Thermal processing is used in semiconductor manufacturing to anneal crystal damage caused by implant and to electrically activate the impurities that were introduced to form devices. The continuing size reduction in devices has greatly limited both the temperature and time available for thermal processing in order to limit the motion of impurities during thermal processing steps. Physical effects that were previously unknown or considered to be second order are becoming the dominant factors in the redistribution of dopants during thermal processing. Consequently, the models for these effects are becoming increasingly more complex, and the semiconductor industry is relying more on simulation to accurately predict the movement of dopants during processing.

The traditional approaches to the implementation of semiconductor diffusion models in simulation programs have not scaled well with the increasing complexity of these models. A new approach, designed to support rapid prototyping of equation-based diffusion models, has been implemented in the ALAMODE simulation program. This approach is based on a *dial-an-operator* paradigm in which simulation models are built from libraries of reusable components and eliminates the burdens of discretization and linearization for the model developer.

The requirements for building an implementation based on this approach are discussed, with particular attention to representation of an equation-based model, robust methods for numerical discretization, design issues specific to simulation software, and efficiency of the implementation. Simulation results are presented for a representative set of semiconductor diffusion models that have been scripted for ALAMODE, including models with fully kinetic treatment of extended defects.

Acknowledgements

There have been a great number of people who have contributed to my research life at Stanford. I am glad to express my gratitude to my advisor, Professor Robert W. Dutton, for his many years of support and encouragement. He has cultivated a diverse and complimentary environment that is fertile ground for novel research in semiconductor and CAD areas.

I would like to thank the other members of my dissertation committee, Professor Kincho Law and Professor Peter Pinsky, for their advice, criticism, and recommendations.

Many other people contributed to the development of this research and to the ALAMODE program. At Stanford, Dr. Ronald Goossens and Dr. Edwin Kan provided direction in the early stages. Martin Gander helped with the coding and was an invaluable resource on numerical analysis. Tao Chen provided an interface to the CAMINO mesh generator. Dr. Peter Griffin's accessibility and expertise in the area of physical models greatly facilitated development of model scripts. Dr. Martin Giles of Intel extensively tested the code and was a driving force behind many useful features and the overall robustness of the code. Alp Gencer of Boston University provided many ready to run diffusion model scripts.

I would also like to thank Fely Barerra and Maria Perea for their help with administrative issues.

I am grateful to the members of the TCADre for their interactions, both research and social. In addition to those mentioned above, I would like to acknowledge Zak Sahul, Nathan Wilson, Ken Wang, and Michael Kwong for many fruitful discussions.

I gratefully acknowledge the financial support of DARPA under contracts DABT63-95-C-0090 (ParaSCOPE) and DABT63-93-C-0053 (SPRINTCAD) as well as the Semiconductor Research Corporation under contract SP-101.

Finally, I would like to thank friends and family. I especially thank Jayme for her support, patience, and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The continuing and rapid shrinking of the size of semiconductor devices has ushered in the information revolution by providing inexpensive and increasingly more powerful computers. Continuing at the current rate of progress requires that the design process be kept under control, and the design process, itself, is becoming more complex at a rapid rate. To help keep the design process under control, engineers are increasingly turning toward simulation of both the electrical characteristics of individual devices and also the effects of manufacturing processes used to build devices.

In device simulation, the equations that model the electrical behavior of a device have remained relatively unchanged for years, with the exception of attempts to broaden the applicability of drift-diffusion by using "kitchen-sink" semi-empirical mobility models. In the thermal diffusion area of process simulation, however, models which are developed for the current generation of technology may no longer be adequately predictive to be used as a design aid after only a couple of generations of technology. Physical phenomena that were either unknown or considered to be second- or third-order effects a few generations ago are becoming the dominant mechanisms in dopant migration. The corresponding shrinking of the thermal budget available for annealing has produced a need for an similarly rapid pace in the development and enhancement of models implemented in the simulation tools which aid development of device manufacturing processes.

Traditional means for simulator development have not been able to deliver at the

timely rate required by process engineers. Development of simulation tools within a university environment is usually tied to the lifetime of a graduate student, and may take several years to deliver a usable implementation. The situation is not much better even on the commercial TCAD which can take as much as 3 years before a published model appears in a commercial product [1], and the publication date is often 6-18 months after the model was developed.

At least part of the blame for the long turnaround can be placed on the traditional approach in which models are implemented as hardcoded entities deep in a simulator's source code. This implementation approach requires access to the source code and a sufficient understanding of the internal data structures and the existing implementation paradigms to be able to realize the necessary changes. People with both the physics background necessary to develop models and good software development skills are uncommon, so the implementation of a model is often done by a different person than the one who developed the model. This scenario separates the disciplines of model and simulator development [40]. Ideally, a model developer would identify the mechanisms responsible for the physical phenomena and develop a complete set of equations which describe the model. The simulator developer would then implement the discretized equations and improve the numerics, as needed. Unfortunately, even this scenario is far from optimal. Model specifications that haven't been implemented and tested usually contain errors and are often incomplete, either because the missing pieces are assumed knowledge or because of a simple oversight in the specification. Although the actual coding and testing should uncover most specification and oversight errors, the traditional process to implement model equations in executable code in a simulator is still a slow, error-prone, and manual process.

To provide for rapid prototyping and testing of new models, the model translation process should be automated. Given a formal specification of the model equations, a simulation environment should be able to automatically generate an executable representation that can be used to simulate the model. This research aims to identify the requirements and implementation techniques needed for such an environment.

1.1 An overview of simulator model implementation approaches

Before delving into the design and implementation details needed for a rapid prototyping environment for semiconductor diffusion models, it is useful to look at research approaches used to prototype diffusion model implementations in simulators.

1.1.1 SUPREM and FLOOPS

Both SUPREM [32] and FLOOPS [31] epitomize the traditional approach to TCAD process simulator development. In these tools, selected mainstream models were hardcoded deep within the code. End users could only select the desired model (i.e. the hardcoded system of equations) and provide model parameters, such as the numerical values used to calculate the temperature dependent coefficients in a specific diffusivity model. This is a beneficial interface for process engineers because it provides them with a simple mechanism to select and tune specific models.

Model implementors, however, need access to the source code and an understanding of the data structures and discretization techniques. The effort involved in extending an existing model or in implementing a new model depends critically on how well the changes fit into the existing design abstractions that were used for implementation. The abstractions used for implementation of diffusion models in both SUPREM and FLOOPS were tied to the formulation of very specific sets of diffusion equations, and these abstractions actually hinder implementation of models that do not fit into the limited form of equations derived for those specific models. These design abstractions and how they impact implementation effort are discussed in more detail in Section 4.2.

1.1.2 Finite Element Prototyping Codes

Many finite element codes, such as FEAP [56], provide a programming interface that can be used to insert code that implements different models into the program. In these programs, an equation-based model is discretized and encapsulated in an

"element routine". The primary purpose of the element routine is to generate local element matrices and residuals given the geometric and field data on an element. The element routine is linked with a program that provides the rest of the simulation infrastructure, such as global control, mesh generation and representation, numerical solution algorithms (time integration, nonlinear solution, linear solution, etc.), and postprocessing.

While this approach does provide a common interface for plugging a variety of models into a simulator and most of the underlying infrastructure needed for discretized simulation, the level of abstraction is not very high. Implementation of a set of model equations in an element routine requires significant work because discretization and linearization of the equations are manual tasks.

In addition, FEAP, like many other FEM codes, has built-in paradigms intended to support other simulation domains such as structural mechanics by providing conceptual abstractions for deformation, displacements, and traction boundary conditions. These paradigms are not appropriate for the equations and boundary conditions encountered in semiconductor diffusion models. Attempts to adopt them for use in the implementation of such models would be confusing both to the implementor and to the user of the code.

1.1.3 ZOMBIE

ZOMBIE [28] was developed at the Technical University of Vienna as a general purpose 1D simulation tool targeted at supporting both process and device models. Instead of only providing a fixed set of models, it provided capabilities for end users to implement their own equation-based models which would be compiled and linked with the simulator.

ZOMBIE is implemented as a two-pass system. The first pass scans an input file and generates code which will be compiled and linked to a support library, then executed as the second pass. The library provides support for overall simulation control, grid generation, integration in time, nonlinear stepping, and linear solution.

The model description capabilities are somewhat similar to that of a typical finite

element code and do not insulate a model developer from the need to learn the native language in which the simulator was written or from the intricacies of the mesh and field data structures. The primary advantages of ZOMBIE over a structurally-oriented FEM code is that specification, meshing, and control use paradigms from semiconductor process and device simulation instead of structural mechanics, and the code provides additional infrastructure that is useful for the implementation of process and device models.

To implement a model in ZOMBIE, a developer would need to provide a set of FORTRAN routines implementing the discretization of the coupled system of partial differential equations. The coupled system is assumed to be a system of N equations of the form

$$\sum_{j=1}^{N} T_{ij} \frac{\partial C_j}{\partial t} + \frac{\partial F_i}{\partial x} = G_i \tag{1.1}$$

where T_{ij} provides temporal dependence, C_i are independent variables, F_i are fluxes, and G_i is a net generation. Boundary conditions are described by a constraint which generalizes both Neumann and Dirichlet boundary conditions:

$$\sum_{j=1}^{N} A_{ij} F_j + \sum_{j=1}^{N} B_{ij} C_j = D_i \tag{1.2}$$

A user would minimally need to provide two routines:

1. a flux routine which computes a system-level spatially discretized residual and the corresponding element tangent matrix for the flux terms in the model equations

2. a routine to evaluate T_{ij} and G_i as well as the $\partial G_i / \partial C_j$

If needed, the user also has to supply routines to compute the coefficients for boundary conditions and temperature dependence on parameters.

A significant implementation restriction is a limit of 10 independent variables, and removal of this restriction would require significant modification of the source

code as it is propagated widely in the source code, including names of routines with external linkage. While such restrictions may have been acceptable at a time when computing resources were scarce, such a restriction would prevent implementation of such diffusion models as the fully-kinetic precipitation model [17], which treats each size of {311} clusters[1] independently, often looking at clusters ranging from 2 to 120 atoms.

1.1.4 PEPPER

PEPPER [38] is a 1D silicon process simulator with hardcoded diffusion models similar to those found in SUPREM-III [23] and SUPREM-IV [32]. In addition to these hardcoded models, PEPPER also supports simulation of user-specified models which are described using a "dial-an-operator" mechanism. With this technique, a model implementor specifies equations as a sum of operators which are instantiated from a library of available operators. The operators are essentially templates for discretized terms, and the instances of the operators in an equation that associates the discretization with specific solution variables, thus providing a system-level coupling. The spatial discretization of operators is a 1D linear finite element discretization. The temporal integration engine is LSODI [24], an implicit-form adaptive ODE integrator based on n^{th} order backward difference formula. The LSODI package is used in a mode where the implicit-form Jacobian is computed numerically by differencing residuals, so tangent information need not be coded in the operators.

In addition to the operator library, there is a library of functions that can be used to "plug-in" a limited number of operator expressions. Unfortunately, this capability did not permit the functions in this library to be combined in support of arbitrary user-defined expressions. If a model required an expression that was not present in the library of available functions, a model implementor would have to provide a C routine to compute that expression.

[1] {311} clusters are a specific type of an extended defect that form from excess silicon interstitial atoms in implant damaged silicon wafers. Their dissolution is the primary source of significant enhancements in the apparent diffusivity of impurities during low temperature and/or short lived thermal cycles.

Two other significant limitations are that only one dial-an-operator based model could be defined and simulated at any given time, and the model would only describe diffusion kinetics in a single material region and the immediate boundaries. These limitations prevent the development of diffusion models spanning multiple material regions.

1.2 Outline

The main focus of this research has been to develop a simulation tool that can transform a relatively simple formal specification of a diffusion model into an executable representation. The prototype developement tool, ALAMODE, is a robust and efficient environment for simulating the evolution of impurity profiles on 1D, 2D, or 3D device structures using state-of-the-art continuum diffusion models.

Chapter 2 develops an information model which can be used as a representation of equation-based diffusion models. The information model considers representation of practical physical models which are described by different systems of equations in various material regions and the interfaces between them. The models are built using a "dial-an-operator" approach, building equations from a library of reusable discretized terms and providing a second library of reusable functions that can be used to specify arbitrary expressions.

Chapter 3 discusses the numerical techniques used to discretize both space and time. The standard discretization techniques do not guarantee all of the properties needed to robustly compute a solution. The sources of these problems are discussed, and alterations to the discretization are suggested which will guarantee a robust solution.

Chapter 4 discusses lessons learned related to software design. To design for extensibility, one must be careful when using domain-specific knowledge to develop implementation abstractions and avoid basing abstractions on domain-specific areas that are likely to become nullified. Designing for software integration is also discussed in the context of abstracting interfaces to external software such as a linear equation solver packages and mesh generation utilities. Appropriate use of object-oriented

programming features, such as inheritance, is also discussed.

Chapter 5 discusses the implementation of model evaluation in ALAMODE. Techniques for efficient interpretive evaluation of expressions are presented. Effectiveness of these techniques is assessed from a performance perspective through comparisons with both an on-demand evaluation implementation and equivalent handcoded models in traditional process simulators. Comparisons show that these highly tuned interpreted techniques can be efficient enough to be used in production code implementations.

Chapter 6 presents examples of diffusion, both to show how models are specified and to demonstrate the range of diffusion models that can be implemented. Specific model examples include: a multiple-material two dopant Fermi diffusion model with a segregation flux between materials, a five-species kinetic model for phosphorus diffusion, and a fully-kinetic {311} defect model in which each defect size is solved for independently.

Finally, conclusions and a summary of the major contributions are in Chapter 7. Directions for future work in model specification, discretization techniques, and simulation software integration are also discussed.

Chapter 2

Dial-An-Operator Model Representation

To develop a tool for simulating user-specified semiconductor diffusion models, one must first identify the domain of possible models that will be considered. Although the hierarchy of modeling techniques for semiconductor diffusion includes highly specialized areas such as atomic-level simulation of small numbers of atoms, most current thermal process simulation uses a statistical approach. These statistical models are described as coupled systems of partial differential equations (PDEs). The variables in these PDEs are the concentrations of various species representing impurities, native crystal defects, impurities paired with native crystal defects, and extended crystal defects, as well as secondary variables such as electrostatic potential. Equations in bulk materials describe the migration of species as well as conversions among species due to pairing, dissociation, clustering, and precipitation. Equations at material interfaces describe flux transport across the interface, including trapping and release of impurities in an interface layer and generation and recombination of defects at the interface. In order to simulate the effects of a given diffusion model on the semiconductor device structure for a given set of processing conditions, the PDEs need to be discretized on a mesh, linearized, and integrated in time.

The simulator implementation paradigm used by ALAMODE is based on a *dial-an-operator* approach [19]. The technique seeks to build an efficient data structure

representing the coupled system of PDEs out of a library of reusable components. The library of components implement a discretization that supports dimensionally independent implementation of these components, while maintaining reasonable efficiency for the evaluation of the discretized system of equations. The data structure and the library of components need to be sufficiently rich to represent virtually any PDE-based model for diffusion.

2.1 A representative model for diffusion

To provide a basis for discussion of the objects needed to represent diffusion models, consider the following, a coupled system of PDEs describing an equilibrium diffusion model for two dopants (boron and arsenic) diffusing in silicon and silicon dioxide.

2.1.1 Equations in the bulk silicon

In the bulk silicon, the migration of the dopants is modeled using:

$$\frac{\partial C_{\mathrm{B}}}{\partial t} = \nabla\cdot\left(D_{\mathrm{B}}\nabla C_{\mathrm{B}} - \frac{D_{\mathrm{B}}}{V_T}C_{\mathrm{B}}\nabla\psi\right) \tag{2.1}$$

$$\frac{\partial C_{\mathrm{As}}}{\partial t} = \nabla\cdot\left(D_{\mathrm{As}}\nabla C_{\mathrm{As}} + \frac{D_{\mathrm{As}}}{V_T}C_{\mathrm{As}}\nabla\psi\right) \tag{2.2}$$

where C_{B} and C_{As} are the concentrations of boron and arsenic, respectively; D_{B} and D_{As} are effective diffusivities for each impurity; and ψ is the electrostatic potential. $V_T = kT/q$ is the thermal voltage, k is Boltzmann's constant, q is the electronic charge, and T is the absolute processing temperature.

The diffusivities in the silicon are dependent on the Fermi energy level, and they can be specified by a number of nearly equivalent expressions. One set of expressions for the diffisivities is written in terms of normalized carrier concentration, and is given

as [18]:

$$D_{\mathrm{B}} = D^0_{\mathrm{B}} + D^+_{\mathrm{B}} \left(\frac{n}{n_i}\right)^{-1} \tag{2.3}$$

$$D_{\mathrm{As}} = D^0_{\mathrm{As}} + D^-_{\mathrm{As}} \left(\frac{n}{n_i}\right) \tag{2.4}$$

where the normalized carrier concentration is written in terms of a net active donor concentration ($C_N = C_{\mathrm{As}} - C_{\mathrm{B}}$),

$$\frac{n}{n_i} = \frac{C_N}{2n_i} + \sqrt{\left(\frac{C_N}{2n_i} + 1\right)}. \tag{2.5}$$

The neutral and charged coefficients in the diffusion expression are temperature dependent. The temperature is assumed to be constant throughout the domain.

Another way of writing the diffusivities is to use the Fermi energy level in terms of the potential, relative to the middle of the bandgap, giving

$$D_{\mathrm{B}} = D^0_{\mathrm{B}} + D^+_{\mathrm{B}} \exp\left(-q\psi/kT\right) \tag{2.6}$$

$$D_{\mathrm{As}} = D^0_{\mathrm{As}} + D^-_{\mathrm{As}} \exp\left(q\psi/kT\right) \tag{2.7}$$

As is the case for dopant diffusivities, there are also several alternatives for approximating the potential. One approach is to add the exact Poisson equation to solve for potential as another unknown in the coupled system of equations:

$$\nabla^2\psi = \frac{q}{\epsilon}\left(2n_i \sinh\left(q\psi/kT\right) + C_{\mathrm{B}} - C_{\mathrm{As}}\right). \tag{2.8}$$

A second, more widely used approach is to assume that the analytic solution of the quasi-neutral approximation is sufficient, giving

$$\psi = V_T \mathrm{arcsinh}\left(\frac{C_{\mathrm{As}} - C_{\mathrm{B}}}{2n_i}\right). \tag{2.9}$$

This approach is also less computationally expensive because there are fewer unknowns.

2.1.2 Equations in the bulk silicon dioxide

In the bulk silicon dioxide, a similar system of equations is used but, because silicon dioxide is an insulator, not a semiconductor, there is no active dopant or electrical effects. Hence, the simplified migration equations become:

$$\frac{\partial C_{\mathrm{B}}}{\partial t} = \nabla\cdot\left(D_{\mathrm{B}}\nabla C_{B}\right) \tag{2.10}$$

$$\frac{\partial C_{\mathrm{As}}}{\partial t} = \nabla\cdot\left(D_{\mathrm{As}}\nabla C_{As}\right), \tag{2.11}$$

Unlike the bulk silicon equations, these do not have the electric field term, and the temperature dependent diffusivities are assumed to be spatially independent.

2.1.3 Equations at the interface

At the interface between the silicon and silicon dioxide regions, there is a flux driving the surface concentrations toward their equilibrium segregation ratio. This segregation flux is given by

$$-\hat{n}\cdot D\nabla C = h\left(C_{\mathrm{Si}} - mC_{\mathrm{SiO2}}\right) \tag{2.12}$$

where $\hat{n}$ is the normal pointing from the silicon into the silicon dioxide material, h is a transport factor, and m is the equilibrium segregation coefficient. This simplified interface flux ignores trapping and release of dopants in the interface layer.

2.2 The general form of equations

Although the representative model is rather old and may seem quite primitive compared with current research in thermal diffusion kinetics, the building blocks for the above equations are the same that are encountered in more advanced models. Drawing from the terms in the representative model, bulk and interface equations can be generalized as shown in the following sections.

2.2.1 Generalization of bulk equations

The general continuity equation used for diffusion models in semiconductors is

$$\frac{\partial C}{\partial t} = \nabla\cdot\left(D\nabla C_m\right) \pm \frac{q}{kT}\nabla\cdot\left(DC^{+}\nabla\psi\right) + G - R \tag{2.13}$$

where C is the total concentration of the species (both migrating and non-migrating), C_m is the concentration of the migrating portion, D is the effective diffusivity, C^+ is the charged portion, ψ is the potential, G is the bulk generation rate, and R is the bulk recombination rate. This equation contains a transient term as well as diffusive, advective, and reactive terms, but not all types of terms need to be present. The migrating and electrically active species may either be independent variables in the model or functional transformations of available quantities. Note that all terms in the equation are independent, scalar valued contributions to the equation. The field being solved for is also assumed to be a scalar valued quantity that is both spatially and temporally varying.

An alternative form of the same equation lumps all of the transport fluxes together:

$$\frac{\partial C}{\partial t} = -\nabla\cdot\vec{F} + G - R, \tag{2.14}$$

where $\vec{F}$ is the total flux involving C. Although this form is more compact, it introduces a need for representation of vector quantities instead of maintaining all variables as scalars.

2.2.2 Generalization of interface constraints

Interface constraints take several forms. The form shown in the model above imposes a constraint, based on flux transport between two material regions,

$$\vec{F} = -D\nabla C_m \mp \frac{q}{kT}DC^{+}\nabla\psi \tag{2.15}$$

A flux constraint is assumed to be projected onto the normal to the interface,

yielding another scalar-valued equation, built out of scalar-valued quantities:

$$-\hat{n} \cdot \vec{F} = h\left(\vec{x}, t, C, \frac{\partial C}{\partial t}\right) \tag{2.16}$$

where h is a scalar-valued function which may depend on values immediately on either side of the interface.

Other constraints may be specified as a Dirichlet boundary condition (i.e. fixed value) or as continuity constraints. These are discussed in more detail below.

2.3 An information model for a simulator based on a dial-an-operator equation representation

The above generalizations of the components used to build equation-based models suggest the information model shown by the entity-relationship diagram in Figure 2.1. The style of the diagram is inspired by Shlaer and Mellor [52]. Single instances of objects are represented in boxes, directional arrows indicate relationships among instances and the order of the relationship (to-one or to-many), a "C" is used to indicate a conditional or optional relationships, and a short horizontal bar crossing a line indicates subclass derivation. The definitions of the objects shown on the diagram and many of the relationships are discussed below. The left half of the diagram contains objects used primarily for model representation. The right half of the diagram contains objects used for discretized field representation.

A *Model* is the set of coupled systems of PDEs and boundary/interface constraints describing the transport of species and transformations among species within a domain. A model must be composed of one or more systems, but Dirichlet boundary constraints and field continuity constraints need not be present.

A *System In Region* is a set of coupled equations within a single region. It is composed of a list of equations and is uniquely associated with a single region.

Dirichlet and Continuity Constraints are tuples referencing regions and fields. Dirichlet constraints reference a field in material and a region. Continuity constraints

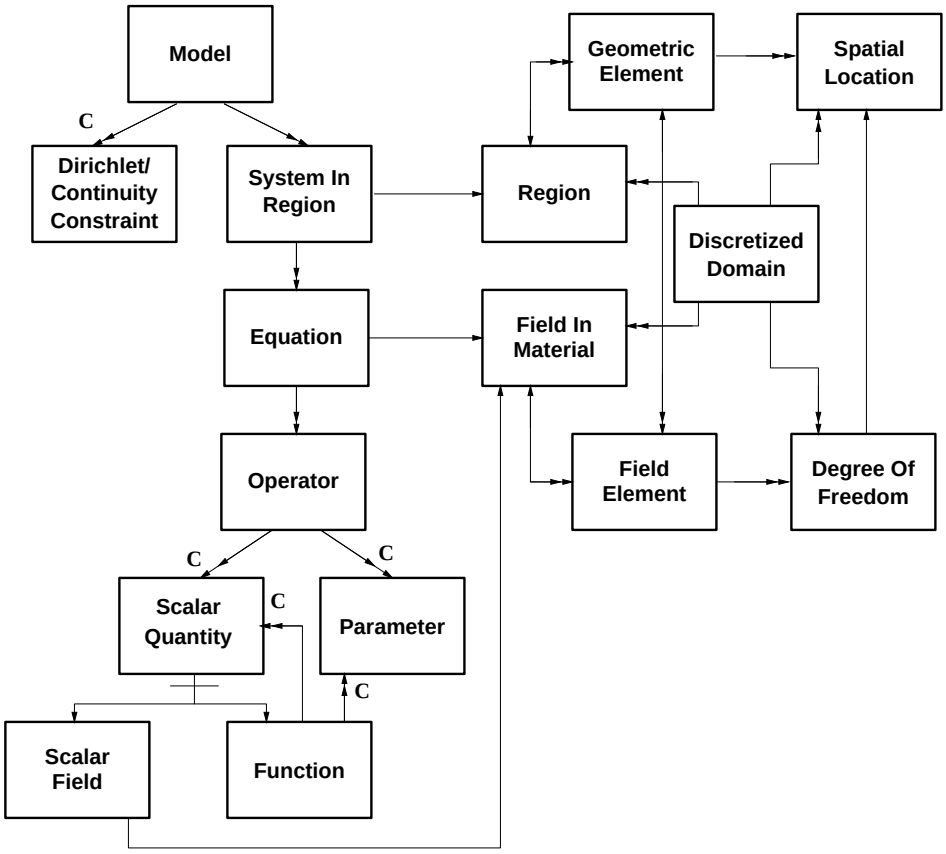

Figure 2.1: Entity-relationship diagram for model and mesh representation

reference a field, two materials, and an interface region.

An *Equation* is a scalar algebraic equation, ordinary differential equation, or partial differential equation. The equation is composed as a sum of independent scalar terms (operators). Each equation is used to solve for a specified field in material.

An *Operator* is a term in an equation. Operators provide the discretization of the term (e.g. a finite element spatial discretization of a nonlinear diffusion term). A particular operator instance will reference scalar quantities or parameters as needed. As an example, for the diffusion operator

$$\nabla \cdot D \nabla C \tag{2.17}$$

the diffusivity could be specified as either a parameter or a scalar quantity and the diffusing concentration would be specified as a scalar quantity. The operator class is an abstract class and is subclassed into child classes that implement different forms of reusable terms (e.g. a diffusion term or a generation/recombination term). The available operators makes up one of the libraries in the dial-an-operator approach.

A *Parameter* is a named scalar value that is not spatially varying.

A *Scalar Quantity* is a spatially varying scalar value. It is subclassed into scalar field and function classes.

A *Function* is a scalar functional transformation on other scalar quantities and/or parameters. As with the operator class, the function class is also an abstract class which is subclassed into child classes that implement different forms of functions populating this second library of reusable components. Examples of functions include basic algebraic components (e.g. sum, difference, product, quotient), traditional functions (e.g. exponential, logarithm), and expressions useful for model development such as those shown in Equations 2.3, 2.4, and 2.5.

A *Discretized Domain* is the collection of regions and fields as well as the components (spatial locations and degrees of freedom) that are building blocks for an unstructured mesh.

A *Region* is a subset of the mesh in which the same system of equations will be applied. Regions are usually materials, material interfaces, and artificial boundaries,

but they may also be used to isolate a subset of a material where the equations differ from those applied elsewhere in the same material (e.g. a damaged or non-crystalline region of a normally crystalline material). A region is tessellated with a mesh of geometric elements.

A *Field In Material* is a named scalar value that is interpolated on the mesh in a region. The reason for specifying this as a field in a material is that most fields are considered to be discontinuous at material interfaces. Thus, three keys are required to sort out the desired value of a field at an interface, the field's name (e.g. Boron), the material (e.g. Silicon), and the location. The field name and the material are combined into this object, providing an unambiguous way to distinguish field values on either side of an interface while maintaining a homogeneous view of a named interpolated scalar value. A field in material is tessellated into a mesh of field elements.

A *Geometric Element* represents a subset of the space spanned by a region and provides some of the geometric properties (e.g. length, area, volume) needed for the discretization. An ordered set of field elements are stacked on a geometric element, one for each field in material that is attached to the region.

A *Field Element* provides for interpolation of values for a field over the subset of space spanned by the corresponding geometric element.

In terms of these objects, the diffusion model presented above would be applied to a discretized domain, partitioned into three regions (silicon, silicon dioxide, and the interface between the silicon and silicon dioxide materials) and four fields in material (BoronInSilicon, BoronInOxide, ArsenicInSilicon, and ArsenicInOxide).

The model would consist of three systems. The system in region for the bulk silicon dioxide would consist of Equation 2.10 solving for the chemical boron concentration in the silicon dioxide, and Equation 2.11, solving for chemical arsenic concentration in the silicon dioxide. Each equation has a first-order transient operator and a single diffusion operator. A parameter would be used for each of the diffusivities in the diffusion operators, and the appropriate field would be specified for the diffusing concentration.

The system in region for the bulk silicon would consist of Equations 2.1 and 2.2,

each of which is composed from three operators (the first-order transient operator, a diffusion operator, and a quasi-neutral electric field operator) and possibly Equation 2.8. The diffusion operators could use diffusivities drawn from the Function Library either of the form given by Equations 2.3 and 2.4 and using a function for the normalized carrier concentration given by Equation 2.5, or of the form given by Equations 2.6 and 2.7. These operators and functions would reference other user-defined functions for net active concentration, user-specified parameters, and user-specified fields to complete the specification of each instance of a function or operator drawn from the Function and Operator libraries.

The system in region for the interface would consist of two equations enforcing the flux condition in Equation 2.12. Each equation would consist of a single operator for the segregation constraint.

2.4 Summary

A simple thermal diffusion model was used to develop a general representation for a class of continuum diffusion models. The representation provides the ability to describe globally coupled equation-based models spanning multiple regions with the ability to specify different physical models in each region. The equations, themselves, are composed using instances of terms that are "dialed in" from an operator library. A second library provides function components in support of representation of arbitrary scalar expressions that can be plugged into the terms or other functions. This model representation paradigm can support the description of virtually any continuum semiconductor diffusion model. Furthermore, by carefully avoiding any hint of discretization in the model representation and using only scalar quantities, the model representation provides a dimensionally-independent description for simulation on 1D, 2D, or 3D structures. Finally, the information model for this representation was intentionally discussed only in terms of the abstract objects into which it has been decomposed. A concrete mechanism, such as a model scripting language, has to be provided as an interface to a end user. The concrete mechanisms that are provided must mirror the objects and relationships present in this information model.

The eventual goal is to simulate the diffusion models; this requires a conversion of the models into a discretized and linearized executable representation. Discretization techniques that provide a robust solution are covered in the next chapter. Techniques to efficiently implement interpretive execution of a discretized system of equations are covered in Chapter 5.

Chapter 3

Discretizations and Numerical Algorithms

A semi-discrete finite element approach is used to discretize the parabolic partial differential equations used for diffusion modeling. The equations are discretized in space using a finite element approach, resulting in a system of nonlinear ordinary differential equations (ODEs). The system is integrated using implicit techniques appropriate for stiff ODEs. The nonlinear system of equations arising from the integration of the ODEs are solved using Newton techniques.

3.1 The semi-discrete finite element discretization

In a semi-discrete finite element approach, space is discretized with a finite element method, resulting in a system of continuous-in-time ODEs. In the following, the technique is illustrated using a multi-dimensional reactive-diffuse initial boundary-value problem (IBVP) using the notational style of Reference [27]. The diffusion coefficient and the reactive term are assumed to be dependent on the impurity concentration.

3.1.1 Derivation

The concentration of the impurity is an unknown function of space and time ($C = C(\vec{x}, t)$) on a domain ($\Omega \times]0, T[$). The strong form (S) of the PDE model involving only a concentration dependent diffusive term is

$$\begin{aligned} C_{,t} &= \nabla \cdot (D(C) \nabla C) + r(C) \\ C(\vec{x}) &= g(\vec{x}, t) \text{ on } \Gamma_g \\ \vec{n} \cdot D(C) \nabla C &= h(\vec{x}, t, C) \text{ on } \Gamma_h \\ C(\vec{x}, t = 0) &= C_0(\vec{x}) \end{aligned} \tag{3.1}$$

Note that it is not necessary to have a Dirichlet boundary ($\Gamma_g = \emptyset$) for well-posedness. Also, for notational simplicity, $D(C)$ will usually be written as D. For the present, assume that the domain and boundary conditions are fixed (i.e. Ω does not change over time nor do the boundaries where Γ_g and Γ_h are applied).

This PDE model will be transformed to an equivalent semi-discrete weak form of the problem after multiplying by a spatially varying weighting function and integrating over the domain (Ω). Two classes of functions will be introduced. The first represents possible solutions (trial functions) for the concentration; the strong form requires that this function space must be restricted by $C(\vec{x}) = g(\vec{x})$ on Γ_g. The second represents allowable functions for the weighting space (test functions), which is restricted by the homogeneous counterpart of the trial space ($w(\vec{x}) = 0$ on Γ_g). Continuity constraints in the weak form will necessitate additional restrictions to these function spaces. The transformation from strong to weak form proceeds as

follows:

$$\begin{aligned}
\int_\Omega wC_{,t}d\Omega &= \int_\Omega \left(w\nabla\cdot(D\nabla C) + r(C)\right) d\Omega \\
&= \int_\Omega w\left(DC_{,i}\right)_{,i} d\Omega + \int_\Omega r(C)d\Omega \\
&= -\int_\Omega w_{,i}DC_{,i}d\Omega + \int_\Omega \left(wDC_{,i}\right)_{,i} d\Omega + \int_\Omega r(C)d\Omega \qquad (3.2)\\
&= -\int_\Omega w_{,i}DC_{,i}d\Omega + \int_\Gamma n_i wDC_{,i}d\Gamma + \int_\Omega r(C)d\Omega \\
&= -\int_\Omega \nabla w\cdot D\nabla C d\Omega + \int_\Gamma w\vec{n}\cdot D\nabla C d\Gamma + \int_\Omega r(C)d\Omega
\end{aligned}$$

Note the use of Einstein notation ($C_{,i}$) for taking derivatives with respect to an arbitrary number of spatial dimensions.

The trial and test function spaces are constrained as a result of this weak form. Because of the first order derivatives in the weak form, both the test and trial spaces must be first order continuous (i.e. C^0) over the domain. Thus, the test and trial spaces are

$$\mathcal{S} = \left\{C\left(\vec{x},t\right) \in C^0\left(\Omega\times]0,T[\right) \mid C\left(\vec{x},t\right) = g\left(\vec{x},t\right) \text{ on } \Gamma_g\right\} \qquad (3.3)$$

$$\mathcal{V} = \left\{w\left(\vec{x}\right) \in C^0\left(\Omega\right) \mid w\left(\vec{x}\right) = 0 \text{ on } \Gamma_g\right\} \qquad (3.4)$$

The weak form (W) of the problem follows directly. The object is to find $C\left(\vec{x},t\right) \in \mathcal{S}$ such that, for all $w\left(\vec{x}\right) \in \mathcal{V}$,

$$\int_\Omega wC_{,t}d\Omega = -\int_\Omega \nabla w\cdot D\nabla C d\Omega + \int_{\Gamma_h} wh\left(\vec{x},t,C\right)d\Gamma + \int_\Omega r(C)d\Omega \qquad (3.5)$$

A mesh is introduced consisting of a set of elements (ϵ) and a set of nodes (η). The mesh is a simple tessellation of the domain (i.e. $\bar{\Omega} = \cup_{e\in\epsilon}\Omega^e$ and $\bar{\Omega}^a \cap \bar{\Omega}^b = \emptyset$ for $\{a,b \in \epsilon \text{ and } a \neq b\}$). The trial and test function spaces are restricted to interpolations on this mesh (i.e. $C\left(\vec{x},t\right) \in \mathcal{S}^h \subset \mathcal{S}$ and $w\left(\vec{x}\right) \in \mathcal{V}^h \subset \mathcal{V}$). Note that introducing this interpolation may violate $\mathcal{S}^h \subset \mathcal{S}$, as the interpolation is not exact, but $\mathcal{S}^h$ is assumed to match $\mathcal{S}$ as closely as possible along the prescribed boundary

to minimize the impact of this violation. The Galerkin FEM approximation (G) of the weak form is now described. Namely, one seeks to find $C^h(\vec{x},t) \in \mathcal{S}^h$ such that, for all $w^h(\vec{x}) \in \mathcal{V}^h$,

$$\int_\Omega w^h C^h_{,t} d\Omega = -\int_\Omega \nabla w^h \cdot D\left(C^h\right) \nabla C^h d\Omega + \int_{\Gamma_h} w^h h\left(\vec{x},t,C^h\right) d\Gamma + \int_\Omega r(C^h) d\Omega \tag{3.6}$$

A set of interpolation functions $(N_A(\vec{(x)}))$ is chosen on the mesh such that the value of a function is 1 at the node it is associated with $(N_A(\vec{x}_A) = 1, A \in \eta)$ and 0 at all other nodes $(N_A(\vec{x}_B) = 0$ for $A \neq B)$. Because the Galerkin approximation must hold for all $w^h(\vec{x}) \in \mathcal{V}^h$, let w^h consist of all linear combinations of functions in $\mathcal{V}^h$. That is,

$$w^h(\vec{x}) = \sum_A w_A N_A(\vec{x}) \tag{3.7}$$

where the w_A's are unconstrained except for the homogeneity constraint on $\mathcal{V}^h$ which requires that $w_A = 0$ for nodes on the Dirichlet boundary $(A \in \eta_g)$. The trial solutions will be drawn from the same basis, leading to

$$C^h(\vec{x},t) = \sum_A c_A(t) N_A(\vec{x}) \tag{3.8}$$

where $c_A(t) = g(\vec{x}_A,t)$ for nodes on the Dirichlet boundary. Finally, note that this set of basis functions is only spatially varying, which leads to the following form for $C^h_{,t}$:

$$C^h_{,t}(\vec{x},t) = \sum_A \dot{c}_A(t) N_A(\vec{x}) \tag{3.9}$$

Replacing the appropriate functions in (G) will produce a nonlinear system of

ordinary differential equations.

$$\begin{aligned}\int_\Omega \sum_A w_A N_A \sum_B \dot{c}_B N_B d\Omega &= -\int_\Omega \sum_A w_A \nabla N_A \cdot D\left(C^h\right) \sum_B c_B \nabla N_B d\Omega + \\ &\int_{\Gamma_h} \sum_A w_A N_A h\left(\vec{x}, t, C^h\right) d\Gamma \end{aligned} \tag{3.10}$$

Linearity of the integration allows an exchange of the integration operator with the summation operator, giving

$$\begin{aligned}\sum_A w_A \int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega &= -\sum_A w_A \int_\Omega \nabla N_A \cdot D\left(C^h\right) \sum_B c_B \nabla N_B d\Omega + \\ &\sum_A w_A \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma + \int_\Omega r(C^h) d\Omega \end{aligned} \tag{3.11}$$

or

$$\sum_A w_A \underbrace{\left(\int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega - R_A \right)}_{G_A} = 0 \tag{3.12}$$

where

$$R_A = -\int_\Omega \nabla N_A \cdot D\left(C^h\right) \sum_B c_B \nabla N_B d\Omega + \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma + \int_\Omega r(C^h) d\Omega \tag{3.13}$$

Because the w_A's are unconstrained (for $A \in \eta - \eta_g$), each corresponding equation, G_A, must be 0.

$$G_A = \int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega - R_A = 0 \tag{3.14}$$

for $A \in \eta - \eta_g$.

This system of nonlinear, first-order, ordinary differential equations can be rewritten in a matrix form as:

$$\mathbf{M}\dot{\mathbf{c}} = \mathbf{R}(\mathbf{c}) \tag{3.15}$$

where $\mathbf{c} = \{c_B\}$ and $\mathbf{R} = \{R_A\}$. The matrix ($\mathbf{M}$) is usually referred to as the mass matrix, following a structural dynamics interpretation. These equations can be integrated in time using any of a number of ODE techniques.

Although this derivation only considers a single reactive-diffusive equation, it is easily applied to systems of reactive-diffusive equations by discretizing each partial differential equation individually and assembling each discretized equation into a global system similar in form to (3.15). For the general system of reactive-diffusive PDEs,

$$\begin{aligned} A_1 C_{1,t} &= \nabla\cdot D(C_1, C_2, \ldots, C_n)\nabla C_1 + r(C_1, C_2, \ldots, C_n) \\ A_2 C_{2,t} &= \nabla\cdot D(C_1, C_2, \ldots, C_n)\nabla C_2 + r(C_1, C_2, \ldots, C_n) \\ &\vdots \\ A_n C_{n,t} &= \nabla\cdot D(C_1, C_2, \ldots, C_n)\nabla C_n + r(C_1, C_2, \ldots, C_n) \end{aligned} \tag{3.16}$$

where A_k is either 0 or 1, depending on whether the equation has a first-order time term, the resulting assembly of discretized equations is

$$\begin{bmatrix} \mathbf{M}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{22} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{M}_{nn} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{C}}_1 \\ \dot{\mathbf{C}}_2 \\ \vdots \\ \dot{\mathbf{C}}_n \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1(\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_n}) \\ \mathbf{R}_2(\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_n}) \\ \vdots \\ \mathbf{R}_n(\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_n}) \end{bmatrix}, \tag{3.17}$$

where $\mathbf{M}_{kk}$ is the mass matrix obtained by discretizing equation k, $\mathbf{C}_k$ is the vector of unknowns associated with the field variable C_k and $\mathbf{R}_k$ is the residual vector from the discretization of equation k. This can be cast into the same matrix form as (3.15),

$$\mathbf{M}'\dot{\mathbf{c}}' = \mathbf{R}'(\mathbf{c}'). \tag{3.18}$$

The semi-discrete FEM discretization obtained above was developed in a *variationally consistent* fashion (i.e. without any ad hoc treatment of discretization of any particular term or the modification of mass matrix). The main advantage of this variationally consistent discretization is that it provides optimal convergence rates [27] for both spatial refinement (commonly called h-refinement) and in the order of the polynomials used as the basis functions (commonly called P-refinement). Unfortunately, as will be demonstrated in the next section, the variationally consistent discretization does not preserve certain properties of the original continuum system and will not provide a robust discretization for solution of arbitrary systems of reactive-diffusive equations on domains with poor quality mesh.

3.1.2 Robust semi-discrete FEM

The chief failure of the variationally consistent discretization is that it does not guarantee that concentration variables will remain positive. This is generally not a problem for "academic" purely diffusive systems, but it is a serious problem if one wishes to provide a robust solution to practical reactive-diffusive systems, especially those encountered in semiconductor diffusion models. The reason why the discretized solution must maintain the property that concentration variables remain positive can easily be demonstrated by examining the solution to the following system of equations which contains terms that model the annihilation/generation of native point defects.

$$\frac{\partial I}{\partial t} = \cdots - K_r(IV - I^*V^*) \tag{3.19}$$

$$\frac{\partial V}{\partial t} = \cdots - K_r(IV - I^*V^*) \tag{3.20}$$

$$\tag{3.21}$$

K_r is a positive constant. If both I and V somehow become negative and their product (IV) is larger in magnitude than I^*V^*, then both I and V will continue to be driven further negative. If there is no other significant influence in the equations, both I and V will exponentially grow in magnitude over time. This behavior is neither correct nor stable.

Yeager [57] formally analyzed nonlinear convergence properties of discretized systems of transient reactive-diffusive equations and produced similar requirements on the discretized solution. In order to guarantee nonlinear convergence, the reactive portion of each equation must be *reactive-definite*. That is, the sum of reaction terms, $r(\cdot)$ must:

1. have a locus, S, contained within D_R (the domain of $r(\cdot)$), such that $r(S) = 0$, and

2. $\partial r(\cdot)/\partial C_k < 0$ for all k and all C_k in D_R.

The first requirement guarantees that the reaction process has locus of attainable equilibrium points, and the second requirement guarantees that the locus is stable. If these conditions are met by the user-specified system of equations, and if the solution produced by the discretization does not violate these conditions, then nonlinear convergence is guaranteed for a Newton approach with only an upper bound on the time-step. Note that negative concentrations would violate the second condition for the annihilation/generation reaction ($K_r(IV - I^*V^*)$).

Thus, to ensure a robust discretized solution of well-posed systems of reactive-diffusive equations, such as those derived from semiconductor diffusion models, one must ensure that concentrations will remain positive in the computed solution. Unfortunately, the variationally consistent semi-discrete FEM discretization provides no such guarantee, not even for the time-continuous solution. Although spatial refinement of the mesh could have been used to reduce any error in the computed solution to the point where concentrations would remain positive, preliminary experiments with this approach indicated that the refinement would lead to a prohibitively large number of nodes. Instead, approaches which changed the discretization were investigated. Two changes are needed: 1) the mass and reactive terms must be nodally lumped, and 2) if elements do not meet certain geometric constraints, then an inconsistent, nodally decoupled discretization is required.

Nodal lumping

Strang and Fix [55] mention that a diagonally lumped mass matrix is needed to guarantee the discrete form of maximum/minimum principle (DMP) for transient heat conduction, which is the same equation as pure diffusion. The maximum/minimum principle [58] states that the maximum and minimum values in the solution of linear diffusion equation,

$$\frac{\partial C}{\partial t} = \nabla \cdot D \nabla C, \tag{3.22}$$

on $\Omega \times]0, T[$ will occur either in the initial conditions ($C(\vec{x}, t = 0)$) or at an external boundary of the domain. If the initial values of C are positive, and the problem is well-posed in that the boundary conditions will not drive C negative, then the time-continuous solution, $\mathbf{C}(t)$, computed by a mass-lumped semi-discrete FEM discretization will remain positive. Requiring satisfaction of the DMP is more restrictive than needed to ensure positive concentrations in the computed solution, but without a specific well-posed system of equations, it is difficult to produce less restrictive heuristics.

Ciarlet also investigated the conditions needed to satisfy the DMP from a matrix perspective [12]. Although motivated from a finite difference perspective, the analysis is general enough to apply to semi-discrete FEM of the linear diffusion equation and was applied to the FEM discretization in [3]. The matrix constraints were derived for a generalized trapezoid time integration algorithm with the parameter (θ, $0 \leq \theta \leq 1$) applied to arbitrary mass and diffusion matrices. The case where $\theta = 1$ is equivalent to time continuous case. The solution will satisfy the DMP if all of the following constraints are met:

$$M_{ij} - (1 - \theta)\Delta t K_{ij} \geq 0 \qquad \text{for all } i, j \tag{3.23}$$

$$M_{ij} + \theta \Delta t K_{ij} \leq 0 \qquad \text{for } i \neq j \tag{3.24}$$

$$M_{ii} + \theta \Delta t K_{ii} \geq \sum_{j \neq i} |M_{ij} + \theta \Delta t K_{ij}| \tag{3.25}$$

Although the off-diagonal entries in a variationally consistent mass matrix are typically non-negative, some entries in certain higher order elements may be negative, which would violate the first constraint. For the second constraint to hold for any Δt, the non-negative off-diagonal entries of the mass matrix must be 0. The mass matrix must be a diagonally lumped matrix. Given this condition, a second condition mandated by the second constraint is that the off-diagonal entries of the diffusion matrix must not be positive. This latter condition leads to geometric constraints on each element which will be discussed in the next section.

For a variationally consistent mass matrix, the second criterion suggests that there may be a minimum time-step that satisfies the DMP. The constraint on the minimum time-step is fairly easy to derive for specific meshes. For example, for a uniform 1D mesh with linear elements, the constraint is [47]:

$$\Delta t \geq \frac{1}{6\theta} D h^2. \tag{3.26}$$

It is important to note that this is a constraint on the minimum time-step which is opposite to the usual upper bounds imposed by stability and accuracy concerns. This two-sided constraint is not acceptible because the minimum time-step may also be larger than the time-step required to meet a certain accuracy in the time integration, or even larger than the time integration interval for a transient diffusion simulation on a given mesh. Note also that the minimum time-step is related to a measure of the element size. The lower bound on the time-step could also be reduced by refining the mesh, but that is not attractive from an efficiency viewpoint.

Using a nodally lumped mass matrix eliminates the need for a minimum time-step. There are a number of general approaches to mass lumping. Perhaps the easiest to implement is the row-sum technique, which generates a lumped mass matrix using

$$m^e_{aa} = \int_{\Omega^e} N_a d\Omega \tag{3.27}$$

The danger of this approach is that it does not guarantee positive lumped masses for

certain elements [27]. A second common approach guarantees positive entries

$$m^e_{aa} = \alpha \int_{\Omega^e} N_a^2 d\Omega \tag{3.28}$$

where α is a scaling factor chosen to preserve element mass:

$$\alpha = \frac{\int_{\Omega^e} 1 d\Omega}{\sum_{a=1}^{n_{en}} \int_{\Omega^e} N_a^2 d\Omega}. \tag{3.29}$$

Unfortunately, both of these approaches can produce undesirable artifacts in the computed solution. For 2D and 3D meshes composed of linear simplex elements (triangles with $n_{en} = 3$ and tetrahedra with $n_{en} = 4$), both of the above lumping approaches produce an element mass matrix with entries that are $1/n_{en}$ of the element's area or volume. For a somewhat uniform mesh, the entries in the assembled mass matrix are roughly proportional to the number of elements surrounding the node. For the mesh shown in Figure 3.1(a), the mass entry at node 1 would be twice that of the mass entry at node 2. Hence, the rate of change at these nodes artificially differs by a factor of 2, and that difference would show up in the computed solution. A second undesirable artifact is caused by the mismatch of lumped quantities between the interior and boundary elements at nodes 3 and 4 in Figure 3.1(b). This mismatch leads to an artificial difference in the applied boundary fluxes, and can produce asymmetric results, even for symmetric initial and boundary conditions.

Both of these undesirable solution artifacts can be eliminated by using a control-volume approach to lumping. In this approach, the equivalent of the lumped mass matrix entry for a node is the control volume surrounding a mesh node. For triangle and tetrahedra meshes, the control volume for a given node is determined by the intersection of the Voronoi dual of the mesh with each region's boundary. This is illustrated in Figure 3.2.

Figure 3.3 shows the differences in simulation results for a linear diffusion equation using FEM and control-volume lumping. For FEM lumping, the contours of the computed solution show the jagged effect of the factor of 2 difference in rates between neighboring nodes. Control-volume lumping produces a smooth solution.

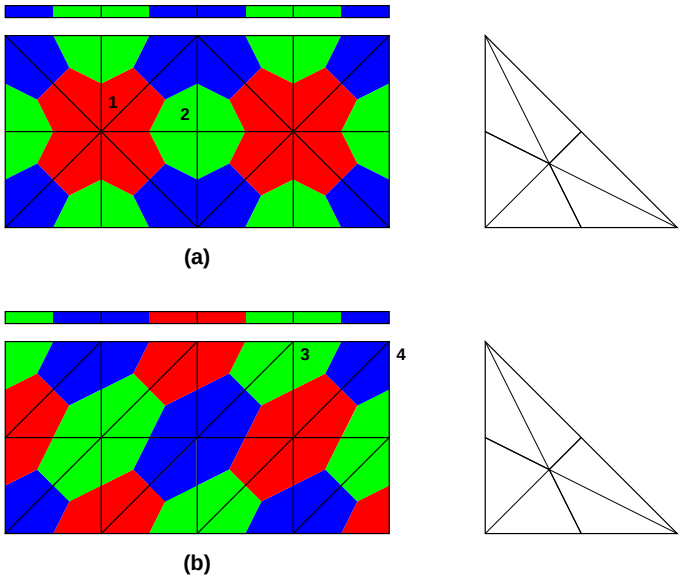

Figure 3.1: Areas associated with the entries of the mass matrix for two uniform triangular meshes.

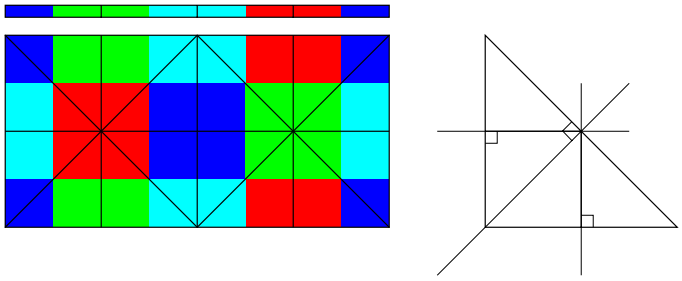

Figure 3.2: Control volumes for nodes in a triangular mesh.

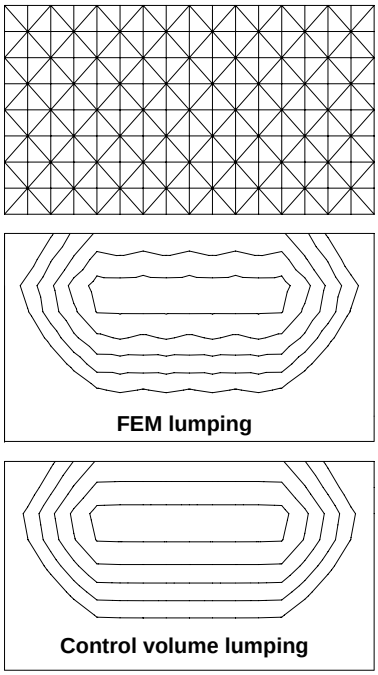

Figure 3.3: Simulated contours for a linear diffusion model using both FEM and control-volume approaches to mass lumping.

Introduction of a lumped mass matrix also brings into question the global conservation of the dopant in the semi-discrete flux discretization in the absence of generation/recombination or external fluxes. Conservation still holds, though not in a strict variational sense as the integral of the concentration over the domain:

$$D_{\text{total}}(t) = \int_{\Omega} C(\vec{x}, t) d\Omega. \tag{3.30}$$

Instead, define the total dopant using

$$D_{\text{total}}(t) = \mathbf{1}^T \mathbf{M}_{\text{L}} \mathbf{C}(t) \tag{3.31}$$

where $\mathbf{1}^T = \{1 \quad 1 \quad \cdots \quad 1\}$.

To test for conservation, differentiate the total concentration,

$$\frac{dD_{\text{total}}(t)}{dt} = \mathbf{1}^T \mathbf{M}_{\text{L}} \dot{\mathbf{C}}(t) \tag{3.32}$$

and use Equation 3.15 for $\dot{\mathbf{C}}(t)$:

$$\begin{aligned} \frac{dD_{\text{total}}(t)}{dt} &= -\mathbf{1}^T \mathbf{M}_{\text{L}} \mathbf{M}_{\text{L}}^{-1} \mathbf{R}\left(\mathbf{C}(t)\right) \\ &= -\mathbf{1}^T \mathbf{R}\left(\mathbf{C}(t)\right) \\ &= \sum_A \int_{\Omega} \nabla N_A \cdot D \nabla N_B d\Omega \\ &= \int_{\Omega} \nabla \underbrace{\sum_A N_A}_{=1} \cdot D \nabla N_B d\Omega \\ &= \int_{\Omega} \nabla 1 \cdot D \nabla N_B d\Omega \\ &= 0 \end{aligned} \tag{3.33}$$

Thus,

$$\frac{dC_{\text{total}}}{dt} = 0. \tag{3.34}$$

The discretization conserves total dopant in this measure.

The final concern is how to discretize the reactive terms. If the reactive terms were discretized in a variationally consistent fashion, then, in the absence of diffusive terms in an equation, the system of ODEs would be

$$M_{AA}\dot{C}_A = \int_\Omega N_A r(\cdot) d\Omega \tag{3.35}$$

Clearly, this would add a spatial influence that is not present in the original strong form of the equation. Reactive terms should also be nodally lumped, using the same nodal mass measure that is used for the first-order time term.

$$M_{AA}\dot{C}_A = M_{AA} r_A(\cdot) \tag{3.36}$$

3.1.3 Geometric constraints on elements

Nodal lumping, alone, does not guarantee that the solution computed from the flux discretization will satisfy the DMP. From Ciarlet's second constraint (3.24), off-diagonal entries in the diffusion matrix must not be positive. For meshes composed of anything other than linear 1D elements, this cannot be guaranteed for arbitrarily shaped elements. For the discretization of a linear diffusion operator, the worst-case element constraints (i.e. for a single isolated element) are:

- Linear triangles must not contain interior angles that are obtuse [13]. This result is trivially extended to linear tetrahedra because the off-diagonal entries in the element diffusion matrix for either element are proportional to the cosine of the angle between adjacent edges or faces.

- Rectangular bilinear quadrilaterals have a constraint on the aspect ratio of the edge lengths [11]. A simple extension of this analysis shows that the only 3D trilinear "brick" element that satisfies the DMP is a cube.

- Higher order elements have more restrictive geometric constraints than low-order elements. For example, only an equilateral triangle satisfies the DMP among six-noded quadratic triangles [25].

These constraints are overly restrictive for multiple-element meshes because nodal coupling in adjacent elements may compensate for the incorrect sign in an element's diffusion matrix once these element matrices are assembled into the global diffusion matrix. Although relaxed geometric constraints for multiple elements could be developed for the linear diffusion case, in the nonlinear diffusion case, the diffusivity is not constant. A formal analysis considering an arbitrary positive diffusivity function is very difficult. Instead, one can assume that the mesh is sufficiently refined that the diffusivity function on each element can be approximated by a constant effective diffusivity over the element ($D^e(\vec{x},t) \approx \bar{D}^e$ on Ω_e during the time-step). With this assumption, the assembly of the global diffusion matrix scales element (geometric) diffusion matrices by $\bar{D}^e$ for the element. Because there are no constraints on the relative magnitudes of the effective diffusivities of neighboring elements, a neighboring element cannot be counted on to compensate for the incorrect sign, regardless of its geometry. The isolated element geometric constraints must be met to ensure that the DMP will be satisfied for the diffusion discretization.

Although the isolated element geometric constraints can usually be met for 2D triangle meshes, it seems impossible that they will be met for non-uniform 3D tetrahedral meshes. Without requiring spatial refinement, a robustly computed solution can still be obtained if the geometric diffusion matrix is modified to meet the off-diagonal constraint. Any positive off-diagonal components that are discovered in the geometric diffusion matrix should be added to the diagonal entry in the same column and then set to zero. Adding the same column to the diagonal entry preserves the dopant concentration property of the discretization. The drawback of this decoupling correction is that the discretization is no longer consistent, and there is no longer any guarantee of h-convergence with mesh refinement.

Figure 3.4 demonstrates the dangers of decoupling. Laplace's equation ($\nabla^2\Psi = 0$) is solved in a trapezoidal domain using both a consistent flux discretization and the decoupled flux discretization. The boundary conditions include both Dirichlet values on the top and bottom edges of the domain and a zero-flux Neumann constraint on the left and right edges. The zero-flux boundaries would produce contours of Ψ that are perpendicular to left and right edges. The skewed mesh was uniformly refined, with

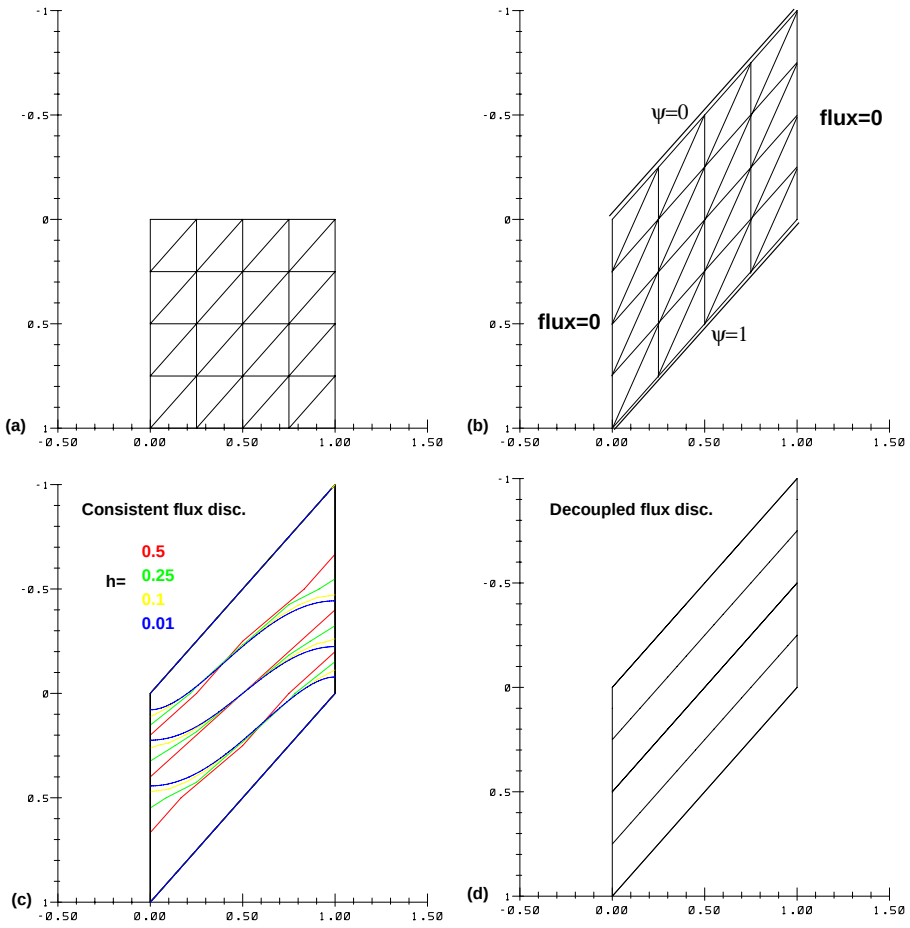

Figure 3.4: Effects of decoupling on h-convergence: (a) original and (b) skewed meshes. Contours of Ψ for a (c) consistent and (d) decoupled flux discretization.

an obtuse triangulation generated for mesh spacings of 0.5, 0.25, 0.1, and 0.01. The consistent flux discretization does converge to both sets of boundary conditions as the mesh is refined. The solution computed by decoupled flux discretization, however, does not change as the mesh is refined and the solution fails to capture the applied zero-flux boundary condition.

Although this example does demonstrate the dangers of decoupling, it is a contrived example. In real meshes, the decoupling is typically not systematically applied, and the effects of decoupling on the solution would not be as pronounced. In any case, ALAMODE is not alone in utilizing a potentially inconsistent discretization. Other

process simulators, SUPREM and FLOOPS for example, also decouple nodes for their control-volume based flux discretization when certain geometric conditions are not met. Doing so at least allows the simulators to compute a solution, even though the error in the solution may not be bounded by a measure of the mesh spacing.

3.2 Time integration algorithms

The transient solution of (3.15) will be considered next. The initial condition ($\mathbf{c}(t=0) = \mathbf{c}_0$) can be computed from the given $C_0(\vec{x}, t=0)$. As is the case with most ODE integration techniques, time is discretized on intervals given of $\Delta t_n = t_{n+1} - t_n$.

3.2.1 Generalized Trapezoid Family

A generalized trapezoid family of methods [27] is one approach used for time integration. This method consists of the following equations.

$$\mathbf{M}\mathbf{v}_{n+1} = \mathbf{R}(\mathbf{c}_{n+1}) \tag{3.37}$$

$$\mathbf{c}_{n+1} = \mathbf{c}_n + \Delta t_n \mathbf{v}_{n+\alpha} \tag{3.38}$$

$$\mathbf{v}_{n+\alpha} = (1-\alpha)\mathbf{v}_n + \alpha \mathbf{v}_{n+1} \tag{3.39}$$

where $\mathbf{c}_n$ and $\mathbf{v}_n$ are approximations to $\mathbf{c}(t_n)$ and $\dot{\mathbf{c}}(t_n)$, respectively. The parameter α is taken to be in the interval $[0, 1]$. Equations 3.37-3.39 can be solved either for concentrations or rates.

Concentration form

In the concentration implementation, define a predictor for $\mathbf{c}_{n+1}$ as

$$\tilde{\mathbf{c}}_{n+1} = \mathbf{c}_n + (1-\alpha)\Delta t_n \mathbf{v}_n \tag{3.40}$$

and combine this with 3.38 and 3.39 to obtain:

$$\mathbf{c}_{n+1} = \tilde{\mathbf{c}}_{n+1} + \alpha \Delta t_n \mathbf{v}_{n+1} \tag{3.41}$$

or

$$\mathbf{v}_{n+1} = \frac{\mathbf{c}_{n+1} - \tilde{\mathbf{c}}_{n+1}}{\alpha \Delta t_n} \tag{3.42}$$

This equation is substituted into 3.37, resulting in the following algebraic system of nonlinear equations.

$$\frac{1}{\alpha \Delta t_n} \mathbf{M} \left(\mathbf{c}_{n+1} - \tilde{\mathbf{c}}_{n+1} \right) = \mathbf{R} \left(\mathbf{c}_{n+1} \right) \tag{3.43}$$

Thus, the nonlinear system becomes

$$\mathbf{R}' \left(\mathbf{c}_{n+1} \right) = \mathbf{R} \left(\mathbf{c}_{n+1} \right) - \frac{1}{\alpha \Delta t_n} \mathbf{M} \mathbf{c}_{n+1} + \frac{1}{\alpha \Delta t_n} \mathbf{M} \tilde{\mathbf{c}}_{\mathbf{n+1}} = \mathbf{0} \tag{3.44}$$

The rates can be computed by rewriting 3.39,

$$\mathbf{v}_{n+1} = \frac{\mathbf{c}_{n+1} - \tilde{\mathbf{c}}_{n+1}}{\alpha \Delta t_n} \tag{3.45}$$

Rate form

Again, we define a predictor for $\mathbf{c}_{n+1}$ as

$$\tilde{\mathbf{c}}_{n+1} = \mathbf{c}_n + (1 - \alpha) \Delta t_n \mathbf{v}_n \tag{3.46}$$

If this is combined with 3.38 and 3.39, we obtain

$$\mathbf{c}_{n+1} = \tilde{\mathbf{c}}_{n+1} + \alpha \Delta t_n \mathbf{v}_{n+1} \tag{3.47}$$

If this is substituted into 3.37, we obtain an equation that will solve for the rates as follows:.

$$\mathbf{R}'(\mathbf{v}_{n+t}) = \mathbf{R}(\tilde{\mathbf{c}}_{n+1} + \alpha\Delta t_n \mathbf{v}_{n+1}) - \mathbf{M}\mathbf{v}_{n+1} = \mathbf{0} \tag{3.48}$$

Remarks on Generalized Trapezoid

The generalized trapezoid algorithm provides access to three specific time-stepping methods depending on the value of α: forward Euler ($\alpha = 0$), trapezoid rule ($\alpha = \frac{1}{2}$), and backward Euler ($\alpha = 1$).

- Any method with $\alpha < \frac{1}{2}$ is conditionally A-stable; time-steps are constrained by the maximum eigenvalue of the discretized system. Arbitrarily large time-steps will produce an amplification matrix with eigenvalues larger than 1 in magnitude. This can lead to incorrect exponential growth in the solution.
- The trapezoid rule ($\alpha = \frac{1}{2}$) is second order accurate in time. All other values of α produce first order methods.
- Backward Euler ($\alpha = 1$) is the only method that is both A-stable and L-stable. With $\alpha = 1$, the eigenvalues of the amplification matrix are bounded by $[0, 1]$ (L-stable). Other values of α ($\alpha \geq \frac{1}{2}$) can produce amplification matrices with eigenvalues near -1 leading to solutions with oscillatory behavior of the solution for large time-steps.

3.2.2 TR/BDF2

Although backward Euler has properties appropriate for robustly integrating the stiff systems of ODEs arising from semiconductor diffusion models, it is only first-order accurate. The TR/BDF2 method, discussed in [4], provides a second order accurate A-stable and L-stable single step method comprised of a trapezoid step over part of the time-step followed by a second order backward difference step over the remainder of the time step.

Let $0 < \gamma \leq 1$ and $\mathbf{c}_{n+\gamma}$ be the computed solution at $t_n + \gamma\Delta t_n$. The TR sub-step goes from t_n to $t_n + \gamma\Delta t_n$ via

$$\mathbf{R}'(\mathbf{c}_{n+\gamma}) = \mathbf{R}(\mathbf{c}_{n+\gamma}) - \frac{2}{\gamma\Delta t_n}\mathbf{M}\mathbf{c}_{n+\gamma} + \frac{2}{\gamma\Delta t_n}\mathbf{M}\mathbf{c}_n + \mathbf{R}(\mathbf{c}_n) = \mathbf{0} \tag{3.49}$$

The BDF2 sub-step goes from $t_{n+\gamma}$ to $t_{n+1} = t_n + \Delta t_n$ via

$$\begin{aligned}\mathbf{R}'(\mathbf{c}_{n+1}) &= \mathbf{R}(\mathbf{c}_{n+1}) - \frac{(2-\gamma)}{(1-\gamma)\Delta t_n}\mathbf{M}\mathbf{c}_{n+1} + \frac{1}{\gamma(1-\gamma)\Delta t_n}\mathbf{M}\mathbf{c}_{n+\gamma} - \\ &\quad \frac{(1-\gamma)^2}{\gamma(1-\gamma)\Delta t_n}\mathbf{M}\mathbf{c}_n = \mathbf{0}\end{aligned} \tag{3.50}$$

The constant multiplying the truncation error is minimized if $\gamma = 2 - \sqrt{2}$. This value of gamma also produces identical tangent matrices for both the TR and BDF2 steps. There is the opportunity to reuse factorizations of the tangent matrix.

Although the TR/BDF2 method is L-stable, it is not monotonic. Either sub-step may produce negative concentrations in the solution. The conditions that can cause negative concentrations can be identified by applying TR/BDF2 to the following ODE:

$$\frac{\partial C}{\partial t} = -\lambda C \tag{3.51}$$

where $\lambda > 0$. Let $z = \Delta t\lambda$. The TR step is

$$C_{n+\gamma} = K_{\mathrm{TR}}(\gamma, z)C_n \tag{3.52}$$

where $K_{\mathrm{TR}}(\gamma, z) = (2 - \gamma z)/(2 + \gamma z)$ and $-1 < K_{\mathrm{TR}}(\gamma, z) < 1$. The sign change in the TR substep is caused by overshoot from taking a large time-step compared to the oscillatory period computed from the eigenvalues of the system. Although this would seem to severely limit the size of the time-step when integrating stiff systems of ODEs, the limitation does not occur in practical systems because the modes associated with the large eigenvalues decay rapidly.

The BDF2 step is

$$C_{n+1} = K_{\mathrm{BDF2}}(\gamma, z)(C_{n+\gamma} - (1-\gamma)^2 C_n) \tag{3.53}$$

where $K_{\mathrm{BDF2}}(\gamma, z) > 0$. If the concentration has been reduced by a significant fraction over the TR step ($(1-\gamma)^2 C_n > C_{n+\gamma}$), the sign on C will change. With effective error control in the time-step, these conditions seldom occur.

The local truncation error (LTE) can be used to estimate a reasonable next time step. One approach is to use the divided difference formula

$$\tau_{n+1} = 2C\Delta t_n \left| \gamma^{-1}\mathbf{R}\left(\mathbf{c}_n\right) - \gamma^{-1}\left(1-\gamma\right)^{-1}\mathbf{R}\left(\mathbf{c}_{n+\gamma}\right) + \left(1-\gamma\right)^{-1}\mathbf{R}\left(\mathbf{c}_{n+1}\right) \right| \tag{3.54}$$

is used to estimate the LTE. Note that C is the constant multiplying the highest order truncation term in the method

$$C = \frac{-3\gamma^2 + 4\gamma - 2}{12(2-\gamma)} \tag{3.55}$$

The LTE is compared to an allowable error tolerance to produce a RMS measure of the normalized pointwise error. Let

$$\mathbf{q}_{n+1} = \mathbf{M}\mathbf{c}_{n+1} \tag{3.56}$$

and

$$e_{n+1,i} = \epsilon_R |q_{n+1,i}| + \epsilon_A \tag{3.57}$$

where ϵ_R is the relative error tolerance to be met and ϵ_A provides a floor to prevent very small (or even zero) rates in $q_{n+1,i}$ from producing a false large relative error. Then, the RMS measure of the normalized pointwise error is given by

$$r^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\tau_{n+1,i}}{e_{n+1,i}}\right)^2 \tag{3.58}$$

A candidate next time step is chosen as

$$\Delta t^*_{n+1} = \Delta t_n r^{-1/3} \tag{3.59}$$

If the normalized error measure, r, is reasonably small (less than 2), the error in the time-step is assumed to be acceptable and the next time step is chosen to be $\min\left(\Delta t^*_{n+1}, 2\Delta t_n\right)$. If r is too large, the current time-step is repeated with $\Delta t_n = 0.7\Delta {t_{n+1}}^*$. The bound on the increase of a stepsize helps to avoid stepsize oscillations that would occur if the estimated next time-step were too large to control the error in the next integration interval, i.e. the interval would have to be integrated again with smaller time-steps that would meet the error criteria.

A second time-step estimator is based on a LTE estimate computed using a Milne's device [57]. A second TR step is taken from $t = t_n + \gamma\Delta t_n$ to t_{n+1}. The solutions of the BDF2 step and the second TR step are compared and used to estimate the truncation error for the current time-step. In ALAMODE the truncation error is computed as a weighted sum of both the RMS and maximum norms of the pointwise relative error.

$$E = \alpha_{rms} \left\| \frac{C_{TR} - C_{BDF}}{\epsilon_R |C_{BDF}| + \epsilon_A} \right\|_2 + \alpha_{max} \left\| \left| \frac{C_{TR} - C_{BDF}}{\epsilon_R |C_{BDF}| + \epsilon_A} \right| \right\|_\infty \tag{3.60}$$

where α_{rms} and α_{max} are the weighting factors assigned to the RMS and maximum norms of the relative error. If the normalized effective error is too large, the time-step is repeated. Otherwise, a candidate time-step increase is computed using the effective error measure using

$$K_t = \left(\frac{7}{E}\right)^{\frac{1}{3}}. \tag{3.61}$$

If $K_t \leq 1$, then the next time-step is selected as

$${\Delta t_{n+1}}^* = K_t \Delta t_n. \tag{3.62}$$

If $K_t > 1$, then the increase in the next time-step is damped using

$$\Delta t_{n+1}{}^{*} = (\log(K_t) + 1)\Delta t_n. \tag{3.63}$$

Including a contribution from the maximum norm helps to prevent too large of a time-step from being selected, particularly in cases where the surface kinetics are the limiting factor instead of bulk kinetics. In this case, the small relative errors at the larger number of nodes in the bulk produce a small RMS measure, but the maximum norm captures the large relative error at the surface nodes. Without the maximum norm contribution, the simulation of many models with strong surface interactions led to sign changes caused by both overshoot in TR and the reduction condition of BDF2. These negative terms lead to failures in the nonlinear solver that required reducing the time-step and resolving the current time interval. Adding the maximum norm contribution to the effective error eliminated many of these problems, and improved overall run time performance.

3.3 Nonlinear Solution Algorithms

A variant of Newton's method is used to solve the resulting system of nonlinear equations. Newton's method solves a system of nonlinear equations, $\mathbf{R}'(\mathbf{c}) = \mathbf{0}$, by successively applying updates in the direction of

$$\Delta \mathbf{c}^{k+1} = -\left(\mathbf{T}\left(\mathbf{c}^{k}\right)\right)^{-1} \mathbf{R}'\left(\mathbf{c}^{k}\right) \tag{3.64}$$

This involves calculation of a residual, which can be calculated directly from any of the modified residual equations (3.44, 3.48, 3.49, and 3.50), and an approximation to the tangent ($\mathbf{T}$). The best approximation to the tangent is the Frechet derivative,

$$T_{AB} = \frac{\partial R'_A}{\partial c_B} \tag{3.65}$$

For the nonlinear systems given by equations 3.44, 3.49, and 3.50, the tangent matrix is simply the sum of tangent of the static residual, $\mathbf{R}(\mathbf{c})$, and a scaled mass

matrix.

$$T_{AB} = \frac{\partial R_A}{\partial c_B} + sM_{AB} \tag{3.66}$$

3.3.1 Convergence

Convergence of Newton methods can be determined using a variety of approaches. One method measures the magnitude of the residual. The nonlinear method has converged if

$$\frac{\left|\left|\mathbf{R}\left(\mathbf{c}^{k+1}\right)\right|\right|}{||\mathbf{R}\left(\mathbf{c}^{0}\right)||} < \epsilon_R \tag{3.67}$$

or

$$\left|\left|\mathbf{R}\left(\mathbf{c}^{k+1}\right)\right|\right| < \epsilon_A \tag{3.68}$$

Another convergence criteria measures the magnitude of the relative change in the full update. The nonlinear method has converged if

$$\left(\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\Delta \mathbf{c}_{,i}^{k+1}}{\epsilon_R \mathbf{c}_{,i}^{k+1} + \epsilon_A}\right)^2\right)^{\frac{1}{2}} < 1 \tag{3.69}$$

Both of these criteria are required to robustly detect nonlinear convergence. In some highly reactive systems, numerical truncation error from subtracting large, nearly equal values limits the reduction in the residual norm, and convergence could not be detected without the update norm.

3.4 Conclusions

A modified semi-discretized FEM approach to the transient solution of nonlinear reactive-diffusive equations has been developed. In order to robustly compute discretized solutions to arbitrary well-posed systems of PDEs, the discretized solution

must maintain properties of the continuum solution which preserve the well-posedness of the continuum equations. For the semi-discrete FEM, nodal lumping must be used for the "mass" matrix and reaction terms, and mesh elements must conform to geometric constraints. Also, if the elements do not conform to geometric constraints, an inconsistent decoupled discretization can be used to ensure the ability to compute a solution, but it is a solution of questionable accuracy. An alternative to decoupling is to spatially refine the mesh.

The ODEs resulting from the semi-discrete FEM discretization are integrated using single-step methods, such as the generalized trapezoid family and a composite method consisting of alternating application of the trapezoidal rule and second order backward differences. The temporally stiff nature of semiconductor diffusion models requires using implicit methods that are preferably both A-stable and L-stable to allow large time-steps. The size of time-steps is controlled by an estimate of the local trunction error associated with temporal discretization.

Chapter 4

Design

4.1 Introduction

Software engineering attempts to define and refine software development procedures which will "obtain economically software that is reliable and works efficiently" [39]. In the classic "waterfall" life cycle [43] (Figure 4.1), analysis and design phases precede actual coding. In the analysis phase, the requirements for the software system are defined. Among these requirements are functionality, performance, and interfacing. The design phase involves decomposing the requirements into well-defined specification of the software architecture, data relationships, data structures, state diagrams, algorithms, and procedural interfaces to be used during the coding phase. If the resulting design is sufficiently detailed, coding is almost a mechanical process, at least in theory.

The design phase should also consider the latter aspects of the software life cycle, testing and maintenance, specifying coding requirements related to validation and producing a design that is easy to maintain and extend. If the life cycle of the software is expected to be closer to the the spiral model [6], in which the software system undergoes a succession of requirement definition, design, development/test, and evaluation to define further requirements, then attention to design aspects related to maintenance and extensibility are more important. Useful CAD software will continue to be enhanced over its lifetime. However, the life cycle differs from a strict

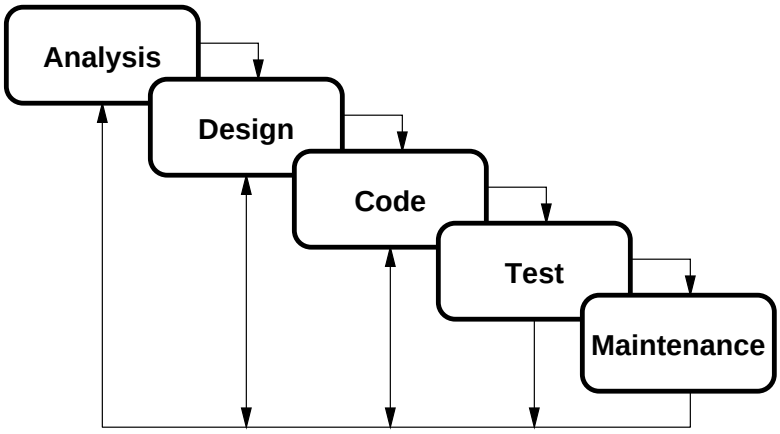

Figure 4.1: Waterfall model of the life-cycle of software

spiral model in that the existing designs and code are not treated as prototypes.

In this chapter, the software design process is discussed with respect to the development of CAD software to be used for simulation of physical processes modeled with systems of PDEs. Special attention is given to how to design software for extensibility and integrability. Specific decomposition approaches, procedural decomposition and object-oriented techniques are discussed along with some of the drawbacks of misusing certain features such as inheritance. Finally, key aspects of the design and implementation of ALAMODE, as a specific case study, are presented.

4.2 Design for extensibility

In TCAD, as in many other CAD domains, a software system is initially developed to simulate certain specific physical processes. The model used to represent the physical process is specified as a system of PDEs along with boundary conditions, and it is often based on assumptions appropriate for a limited range of external conditions. Over time, a physical model may be improved as more knowledge about the physical process is gained or if the external conditions move outside the limited range of applicability. In semiconductor thermal process modeling, the continuing trend towards smaller devices steadily decreases the thermal budget, the temperature

and time, available to manufacture device structures on a wafer. This, in turn, has led to a number a significant changes in the physical models explain OED and TED through the addition of terms and equations solving for point defects and extended defects.

When the physical models change, existing code needs to be altered to reflect changes in the PDEs. In the best case, the code changes are trivial. In the worst case, the changes invalidate assumptions that were used to simplify the original code, and those assumptions are distributed throughout the code. In order to maximize lifetime of physical simulation software and minimize the effort spent on adapting the software to new physical models, the question of how to design the software needs to be addressed. The answer is actually quite simple; one must avoid basing the design on abstractions that are restricted to specific physical models or to a specific physical domain.

Abstraction of the characteristics of physical models and the physical domain is often used to simplify implementation within a simulator. If the abstractions are derived from specific physical models, or if the abstractions contain too many simplifying assumptions about the physical characteristics of the domain which will be simulated, these abstractions can hinder the ability of the code to be modified for use as a simulator of related physical models. Two design abstractions used in SUPREM IV illustrate the dangers of basing abstractions on physical characteristics.

4.2.1 Abstractions based on specific physical models

In SUPREM IV, the fully-coupled diffusion model was the driving force in the simplifying abstractions used within the diffusion portion of the code. In the fully-coupled model, a single equation is used for each impurity, and the pairing reactions are considered to be in a Fermi-level dependent local equilibrium with point defect populations. The impurity diffusion equations were written in a common form. In the semiconductor material, the migration of each impurity is modeled using

$$C_{T,t} = \nabla\cdot\left[D_V C_A \frac{C_V}{C_V^*}\nabla\left(C_A\frac{C_V}{C_V^*}\frac{n}{n_i}\right) + D_I C_A \frac{C_I}{C_I^*}\nabla\left(C_A\frac{C_I}{C_I^*}\frac{n}{n_i}\right)\right] \qquad (4.1)$$

where

C_T	chemical concentration of the impurity
$D_{V,I}$	effective diffusivity of impurity paired with a vacancy or interstitial
C_A	active/mobile concentration of the impurity
$\frac{C_V}{C_V^*}$	supersaturation of vacancies
$\frac{C_I}{C_I^*}$	supersaturation of interstitials
$\frac{n}{n_i}$	normalized electron concentration

In insulating materials,

$$C_{T,t} = \nabla \cdot (D \nabla C_T) \tag{4.2}$$

where the diffusivity is a user-specified, temperature-dependent value. Between material regions, transport of the impurity is given by a flux transport constraint equation, such as Equation 2.12 for a segregation-driven flux.

The implementation of these model equations in SUPREM IV used impurity-specific routines to evaluate each of the following quantities needed for the common form:

- C_A, the active portion of the impurity concentration,
- D, the diffusivity of the impurity,
- the flux boundary conditions appropriate for each material interface and exterior boundary

This design abstraction did provide easy implementation of more advanced models that fit within its paradigm. For example, the deactivation of arsenic with doubly negative charged vacancies, which only requires a change to the routine to compute the active portion of arsenic, effectively uses the same formulation. Any model that changed the assumption of one equation for each impurity, such as a two-stream model

for macroscopic treatment of diffusion in polysilicon grain structures or five-species models that treat the pairing reactions kinetically, could not be easily implemented within SUPREM IV.

4.2.2 Abstractions based on the physical domain

A second physically-motivated abstraction used in SUPREM IV was that the semiconductor processing to be simulated was a silicon technology. This assumption heavily impacted the extension of the SUPREM IV code to also simulate GaAs processes in its incarnation as SUPREM IV.GS [54].

Processing of silicon technology uses only specific materials and impurities. In the implementation of SUPREM IV, specific materials and impurities related to silicon technology were hard-coded using `#define`s. Code developed for physical models that differed between materials used a `switch/case` control structure to handle the fixed number of known materials. Each of these control structures needed to be modified to handle new materials. Similar control structures needed to modified to handle new impurities.

Using untyped `#define`s to internally identify materials and impurities caused a problem with how to uniquely identify silicon, which is a material in silicon technology and an impurity in GaAs technology. In SUPREM IV.GS, impurity silicon is identified as `iSi`, which is inconsistent with the naming convention used for the rest of the impurities.

The most crippling abstraction was that the semiconductor substrate is composed of a single atomic type, silicon, and that only a single atomic type was involved in native point defects. For III-V semiconductors, the substrate is composed of two or more atomic types, and each atomic type can occur as a interstitial or vacancy in the lattice. In tertiary semiconductor materials, such as $Al_xGa_yAs_z$, the chemical composition of the lattice must also be considered. The most accurate models for diffusion in III-IV semiconductors consider the composition and treat the populations of each of native point defects independently. Implementation of these more accurate models was not pursued within SUPREM IV.GS because the assumptions about

a single atomic type was too widespread throughout the code. Instead, the GaAs diffusion models that were implemented were cast within the silicon fully-coupled model, and the model parameters were adjusted for the best fit to limited experimental data.

Basing design abstractions on a specific physical model or domain may facilitate development of models that fit within that domain, but it also has the potential to hinder development of models and adapt the code for use outside of the domain.

4.3 Design for software integration

A second major design concern for CAD software is the integration of the software that implements a physical model with external software packages. The main motivation for this is that a single developer, or even a small team of developers, cannot design and develop quality software that performs every functionality needed in a simulator. External packages need to be integrated, particularly in areas outside of the expertise of the development team, areas such as the solution of sparse linear systems, mesh generation, and visualization. By designing the software to be easily integrated with these external packages, a *plug-and-play* environment can be developed. Different packages for solving a given sparse linear system or generating mesh can be plugged in to benchmark their performance and efficacy when applied to simulation of a specific physical model.

A second motivation to design for integration is that the simulator will not be used solely as a stand alone entity, but will be used as one part of a coupled or sequential process. For example, in TCAD simulation, a process simulator is used to determine the device characteristics, geometry and doping profiles, which will then be fed into a device simulator in order to determine the device's electrical characteristics. In simulating a complete process flow, different applications may be used for each of the different types of process steps, e.g. implant, etch, deposition, and thermal annealing. The representation of the geometrical and field properties of the device structure must be kept consistent during the interchange of data among the applications. The simulator should have an open architecture that allows interfacing to a number of

external data formats.

To realize an implementation that permits easy integration of external software, a software designer has to be knowledgeable of the range of differences in the interfaces for software of like functionality and use this information to define a generic interface that is usable to integrate all external packages using only the defined interface. The routines and data structures in the generic interface can be used wherever needed throughout the code, but any use of "back door" routines, which access functionality available only in a specific external package, should be localized. Examples demonstrating the application of these principles to develop generic interfaces are presented in the following two sections, by means of specific examples using the ALAMODE code.

4.3.1 Integration with sparse linear solvers

ALAMODE implements an interface to a variety of sparse linear solvers. The generic interface is defined by base class in C++, `SparseMatrix`. The `SparseMatrix` class is defined as:

```
// states in which the SparseMatrix object can exist
enum SparseMatrixState {
    Empty,
    LoadingSymbolic,
    Symbolic,
    LoadingReal,
    Loaded,
    LoadedFactored
};

class SparseMatrix {
public:
    // state management
```

```
inline SparseMatrixState getState() const;
const char* stateString() const;
inline int getSize() const;

virtual void resetAndClearMemory() = 0;

// returns true if derived class requires symbolic definition
virtual int needsSymbolicDefinition() = 0;

// Symbolic definition (if needed)
virtual void startSymbolicDefinition( int size ) = 0;
virtual void addSymbolicEntry( int row, int col ) = 0;
virtual void finishSymbolicDefinition() = 0;

// Load of the actual matrix values
virtual void startLoad( int size, int keepPreviousStructure = 0 ) = 0;
virtual void zeroOut() = 0;
virtual void addToEntry( int row, int col, double value ) = 0;
virtual void assignEntry( int row, int col, double value ) = 0;
// various scatter routines to efficiently scatter
// element matrices and diagonal (NOT SHOWN)

virtual void addToDiagonal( const DiagonalMatrix& m ) = 0;
virtual void subtractFromDiagonal( const DiagonalMatrix& m ) = 0;
virtual void scaleBy( double scale ) = 0;
inline SparseMatrix& operator *= ( double scale );
virtual void finishLoad() = 0;

// Once the real data is loaded, we can call these
virtual int solve( const PlainVector& rhs, PlainVector& sol ) = 0;
virtual void multiplyVector( const PlainVector& vin,
```

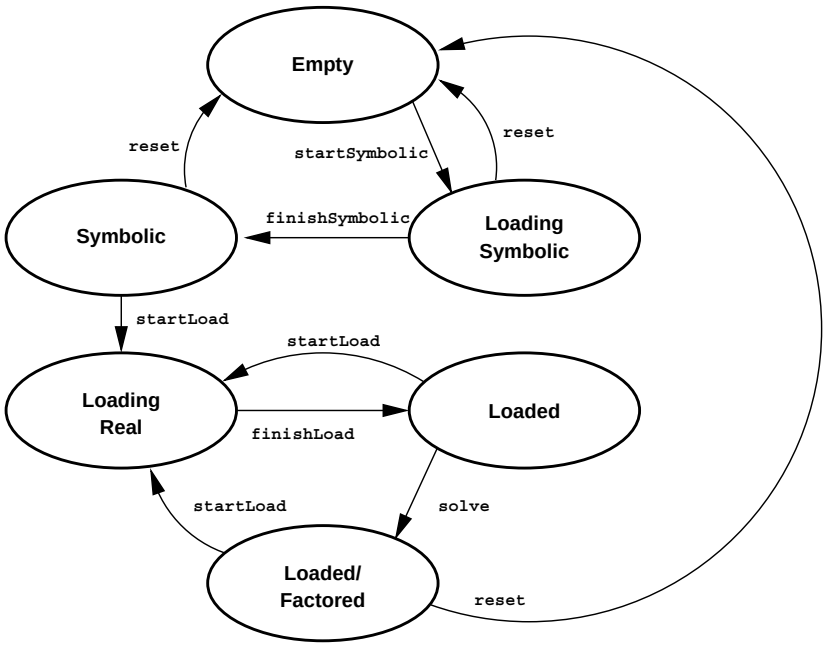

Figure 4.2: States and transitions for the `SparseMatrix` class.

```
                                PlainVector& vout ) = 0;

protected:
    inline void setState( SparseMatrixState state );
    inline void setSize( int size );

private:
    SparseMatrixState state_;
    int size_;

};
```

The `SparseMatrix` class exists in several states as shown in Figure 4.2. The state transition methods can only be called when the `SparseMatrix` is in certain states, as shown on the diagram. Additional state restrictions on methods are shown in

Table 4.1: Valid states where methods can be invoked for a `SparseMatrix` object.

State	Method
`LoadingSymbolic`	`addSymbolicEntry`

`LoadingReal`	`zeroOut`
	`addToEntry`
	`assignEntry`
	`scatter` routines
	`addToDiagonal`
	`subtractFromDiagonal`
	`scaleBy`

`Loaded/LoadedFactored`	`solve` `multiplyVector`

Table 4.1.

The `SparseMatrix` class is not intended to be a general purpose interface for defining and solving any sparse linear system; it is designed to be a buffer between existing general purpose packages that solve sparse linear systems and the code in ALAMODE that needs to access a sparse matrix, namely the core element-based assembly code, nonlinear solution algorithms and time-stepping algorithms. The methods to be supplied by subclasses, particularly the matrix entry scatter methods, are determined by the needs, both data structure and efficiency, of the code in ALAMODE which evaluates element matrices and submatrices. The methods defined by the `SparseMatrix` class are the only routines that are visible for matrix definition and solution in ALAMODE. Thus, any derived class of `SparseMatrix` can be plugged into ALAMODE without modification to the code that makes use of the `SparseMatrix` interface.

Specific packages for linear solution may still require additional methods that are specific only to the package. For example, the derived class that implements the interface to PETSc/SLES [2] preconditioned iterative linear solver, includes additional methods to select the accuracy needed to break out of the iteration, the ILU fill parameter, which iteration to use, and which preconditioner to use. These methods can only be invoked by code that is in the scope of the `SLESMatrix` class, and the

visibility of this class is limited only to where it is required for instantiation and configuration.

```
# Preconditioning methods available in PETSc/SLES
typedef enum {
    SU_SVLU, SU_SVJACOBI, SU_SVSSOR, SU_SVILU, SU_SVBLOCKJACOBI,
    SU_SVICCJP, SU_SVBDD, SU_SVOSM, SU_SVNOPRE, SU_SVICC
} SU_SVMETHOD;
# Linear solution iterations available in PETSc/SLES
typedef enum {
    SU_ITRICHARDSON, SU_ITCHEBYCHEV, SU_ITCG, SU_ITGMRES,
    SU_ITCGSPLIT, SU_ITTCQMR, SU_ITBCGS, SU_ITCGS, SU_ITTFQMR,
    SU_ITLSQR, SU_ITPREONLY, SU_ITCR
} SU_ITMETHOD;

class SLESMatrix : public SparseMatrix {
public:

    inline double getAccuracy() const;
    inline void setAccuracy( double accuracy );

    inline int  getFill () const;
    inline void setFill ( int  fill  );

    inline SU_ITMETHOD getIterationMethod() const;
    inline void setIterationMethod( SU_ITMETHOD iteration );

    inline SU_SVMETHOD getPreconditioning() const;
    inline void setPreconditioning( SU_SVMETHOD preconditioning );
};
```

4.3.2 Integration with external mesh and field data

The need for integration of a number of different processing tools while maintaining a consistent view of the wafer state (geometry, mesh, and field information) has long been recognized. Both References [10] and [50] cover the history of integration efforts and discuss experiences in defining and implementing API interfaces, which support wafer state interchange. Although these API interfaces, such as SWR [20], had potential to permanently solve this integration problem, commercialization efforts failed, and TCAD tools continue to use their own formats. Thus, the problem of integration remains.

A wafer representation is a critical building block for any TCAD tool, but a large list of restrictive requirements on mesh and field is not necessary for interchange with ALAMODE. Following the model representation originally introduced in Figure 2.1 and Chapter 2, the only requirements for mesh and field data are:

- The mesh is partitioned into a number of *regions* with each region supporting a homogeneous system of equations. The usual partitioning will be *materials* and material *interfaces*. Regions are symbolically named, and the names provide a key in the model representation. To provide a more precise capability for selection of the region in which to apply a model, each instance of a material or interface region may also have an instance name associated with it.

- The mesh supports an arbitrary number of interpolated scalar quantities (*fields*). These are also symbolically named, and the names provide a key into the model representation.

- All interpolated quantities are considered to be discontinuous across material interfaces (i.e. one value for each solution for each material at a point). Figure 4.3 shows a partitioning of a mesh into three regions. In this figure, there is one unique scalar solution value for each pair (a-b and c-d) of element's nodes on either side of the interface. Another way of stating this requirement is that there is one value for each field and for each material at a mesh point.

Again, having a well-defined generic interface between software components and

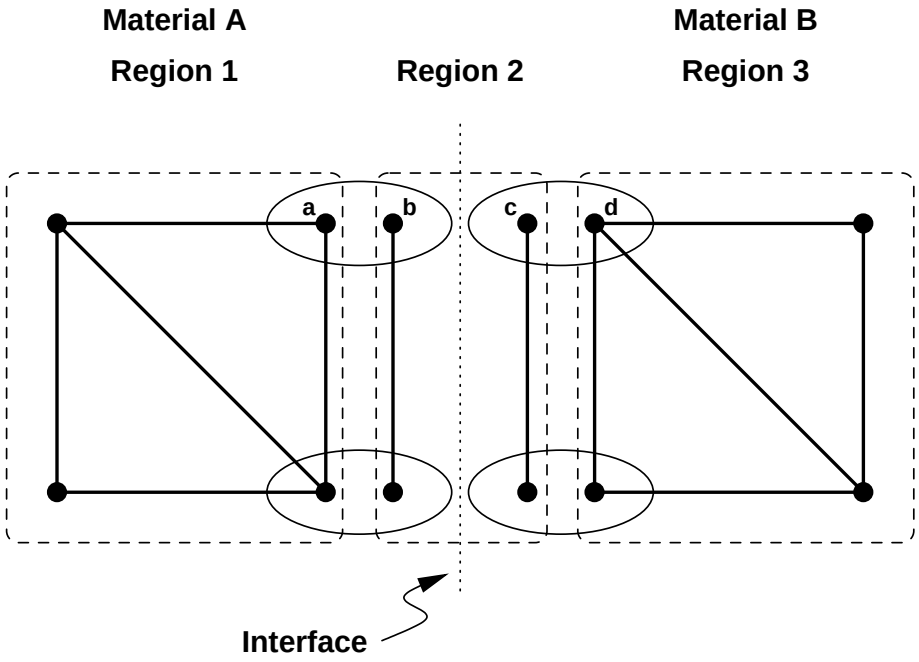

Figure 4.3: Partitioning of a mesh and representation of discontinuity at a boundary

using only what is supported by the generic interface is one requirement for wide integrability. The interface in ALAMODE is based on class interfaces developed to represent a `Domain`. The relevant portions of the class declarations are:

```
class Domain {
public:
    void beginLoadingFieldsAndRegions();
    void endLoadingFieldsAndRegions();

    void beginLoadingElementData();
    void endLoadingElementData();

    void clearAll ();
    void clearElements();
```

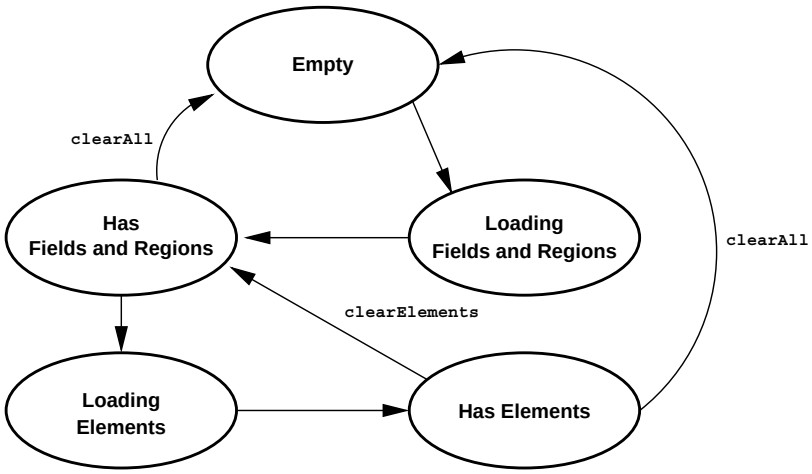

Figure 4.4: States and transitions for the `Domain` class.

```
    RegionID createRegion( const char* regionName );
    FieldID  createField ( const char* fieldName );
    FieldInRegionID attachFieldToRegion( FieldID& field, RegionID& region );

    SpatialPointID addSpatialPoint( SpatialPoint& sppt );
    DegreeOfFreedomID addDegreeOfFreedom( DegreeOfFreedom& dof );
     ...
};
```

The state transitions for the `Domain` class are shown in Figure 4.4. This set of states is motivated by a desire to integrate grid generators that can perform adaptation during the solution of a model. With these state transitions, the equation-based model can remain intact while only the mesh and field data are replaced.

Two other classes, `Region` and `Field`, maintain the actual collections of elements. Each of these classes has a state model consisting of `Empty`, `Loading`, and `Loaded` states. The relevant portions of these class definitions are:

```
class Region {
public:
```

```
    void beginLoadingElements();
    void endLoadingElements();

    void clearElements();

    HECID createHEC( const char* hecName );
    GeometricElementID addGeometricElement( GeometricElement& e,
                                            HECID& hec );
     ...
};

class Field {
public:
    void beginLoadingElements();
    void endLoadingElements();

    void clearElements();

    FieldElementID addFieldElement( FieldElement& e );
     ...
};
```

To initialize a mesh within ALAMODE, one needs to

1. instantiate a domain
2. `beginLoadingFieldsAndRegions`
3. instantiate regions and fields
4. associate fields with regions (`attachFieldToRegion`)
5. `endLoadingFieldsAndRegions`
6. `beginLoadingElementData`

7. create spatial points and degrees of freedom
8. create field elements and geometric elements using the `ID`s created in the previous step and appropriate field and geometric element transforms
9. add each field element to the appropriate field (`addFieldElement`)
10. stack the field elements on top of the corresponding geometric element
11. add each geometric element to the appropriate homogeneous element collection in the appropriate region
12. `endLoadingElements`

Creation of the field and geometric elements is performed using one of the large number of element transforms available in ALAMODE to support the most common element types encountered in TCAD tools. Each of these transforms contains a method to create an instance of a `GeometricElement` or `FieldElement` with the appropriate transform. The design of the transform library is discussed in more detail in Section 4.4.2.

Different types of `ID`s are used extensively in the implementation of ALAMODE. Although these are conceptually interchangeable with an index or a pointer, they provide a type-safe mechanism for access of data and insure that the correct collection or table is accessed. Thus, there is virtually no chance of a programming error such as inadvertently using a region index to access a specific instance of a field or element.

Regions are actually further decomposed than is indicated by Figure 2.1, at least for the purposes of storing elements. Each region contains one or more *homogeneous element collections* (HEC). A separate `HEC` is needed for each set of elements that has a differently ordered set of geometric and field element transforms. This guarantees that within a `HEC`, the size and structure of the element matrix and residual will be fixed for a given system of equations. As an example of a region that needs more than one `HEC`, consider an interface region for a mesh containing prism elements. The elements in an the region can be composed of triangular and quadrilateral 3D surface elements.

Finally, it should be noted that the core of ALAMODE is not designed for mesh generation or as a mesh manipulation tool. With respect to mesh and field data, ALAMODE uses only what is provided. No attempts are made to alter connectivity, fix "bad" elements, or perform mesh refinement within the core mesh and field data structures. The only data that can be changed via solution of the equations are the nodal values, and this is the only data that needs to be transferred back to a mesh interface after a simulation is run.

4.3.3 Practical integration challenges

Two practical challenges related to integrability are: 1) difficulties in interfacing code written in different languages and 2) avoiding name-space conflicts between separate packages.

ALAMODE is implemented primarily in C++ and interfaces to packages written in C++, C, and FORTRAN. The C++ language definition specifies compatibility with C data layouts, how to call external C routines, and how to define a C++ routine so that it can be called from a C program. Unfortunately, it is not always possible to use the C header files from an existing package in a C++ environment. The most common problems encountered are pre-ANSI C function declarations which do not prototype arguments and conflicts with new keywords in C++, such as `new`, `delete`, `class`, and `operator`. This problem can be addressed without altering the header by defining intermediate C routines which eliminate the need for direct inclusion of the incompatible header.

Calling FORTRAN subroutines from C++ or C is also widely supported, provided that a few conditions are met, all parameters are either an integer type or floating point type and data does not need to be shared with the calling program via FORTRAN common blocks. Issues to be aware of include:

- The external linkage name of the FORTRAN routine may differ across compilation environments. In some environments, the FORTRAN name is unchanged. In others, an underscore is appended to the name, or the case of characters in the name is modified to all lower case or all upper case.

- Arrays in C are zero-based and arrays in FORTRAN are one-based.
- Multidimensional arrays in C are laid out in memory in row-major order. In FORTRAN, multidimensional arrays are laid out in column-major order.
- The appropriate FORTRAN runtime support libraries must be linked with the code.

Namespace conflicts are another potential barrier to software integration. Two or more packages may use the same name for a globally visible routine name, data object, or data type. There was one such conflict in the integration of FOREST, a 2D quadtree-based gridding tool [50], to ALAMODE. Each defined a `Region` class, to be used for distinctly different purposes. This name-space conflict was addressed with virtually no modification of either package by adding a header file that renamed each of ALAMODE's class names, adding a hopefully unique prefix:

```
#define Region          AlamodeRegion
```

The only place where an explicit distinction between the two `Region` classes is needed in interface between FOREST and ALAMODE's classes is referred to as `AlamodeRegion`. All other code in either FOREST or ALAMODE continues to use the overly generic class name, `Region`. To avoid namespace conflicts, each package should, at least, avoid using common names, particularly in externally visible interfaces, or should include a package-specific prefix as part of all externally visible names.

4.4 Design methodologies – a comparative analysis

Software engineering proponents encourage following well-defined design methodologies which specify how to decompose a software task and how to document the design. Although there is significant disagreement about the merits and benefits of the details involved with specific design methods, the methodologies can be divided into three

categories: top-down structured design, data-driven design, and object-oriented design [53]. Top-down structured design applies algorithmic decomposition, breaking each complex process down into simple tasks, which will be implemented in separate modules. It is a procedure-centered design, and the data manipulated by these procedures become a secondary concern. Implementation of software designed with a procedure-centered method is aided by support for functions and subroutines found in languages such as ALGOL and FORTRAN. Experience has shown that top-down structured design methods do not scale well with increasing complexity.

Most scientific software, particularly discretized PDE solvers, is largely algorithmic in nature and has traditionally been designed using a top-down structured method, and implemented in a language with very limited data abstraction features such as FORTRAN. The design technique results in a procedural (task-oriented) decomposition. Data and the relationships among data entities are treated as a secondary concern, if at all. While this approach was successful for early generations of primarily 1D TCAD software, procedural decomposition wasn't able to address the increasing complexity of data in multidimensional TCAD software.

Data-driven design [43] concentrates on relationships among data and the mapping of data from inputs to outputs. It has primarily been applied to software designed for two specific areas: 1) information management, such as database manipulation and report generation in the business community, and 2) real-time software, and is most applicable to the design of software systems in these areas. Implementation of software designed with a data-driven method is aided by support for pattern matching, rules, and queries found in languages such as PROLOG and SQL. Although data-centered, simplification via decomposition is virtually nonexistent, and methods in this category also do not scale well with increasing complexity.

The newest category, object-oriented design [8], decomposes the system into collections of cooperating objects. Relationships among entities, including hierarchy and abstraction of entities are employed in selecting the classes of objects to be implemented. State transformations determine the procedures that will be provided by each class. Implementation of software designed with object-oriented methods is

aided by support for encapsulation, class inheritance, and polymorphism found in languages such as SMALLTALK and C++. Design methods in this category scale best with increasing complexity and are more resilient to change and further evolution.

Both abstraction and hierarchy are important in applying object-oriented techniques to software design. In numerical software, however, it seems that too many implementations use hierarchical types (inheritance) to provide extensibility in the models to be implemented. The problem is that inheritance organizes domain-specific knowledge into a hierarchy. FLOODS [33] is used here as an example of a code that develops domain-specific physical model hierarchies using inheritance.

4.4.1 Inheritance in FLOODS

The class hierarchy for PDEs in FLOODS is shown in Figure 4.5. The base class for all PDEs is `PDEinMaterial`, which provides an interface to the `Assemble` method which consumes elements and produces discretized element residual vectors and tangent matrices. The `PoissoninMaterial` subclass adds two methods, `Charge` and `ElectricField`, which assemble the charge and electric field terms in the Poisson equation, assuming an insulating material. A third subclass, `PoissoninSemiconductor`, is able to reuse the `ElectricField` method from `PoissoninMaterial`, but provides its own `Charge` method because the expressions used for the charge terms are different semiconductors than in insulating materials. This class hierarchy is successful in reuse, both of the shared `ElectricField` routine and in the ability to reuse the `PoissoninMaterial` and `PoissoninSemiconductor` classes as object instances for insulating and semiconductor materials where the equations they implement are valid.

However, trying to apply similar class hierarchies to the variety of equations found in models for thermal diffusion quickly runs into extensibility problems. The same physical model limitations on extensibility that exist in SUPREM IV are also present in FLOOPS [31], a process simulator build on the same code base a FLOODS. In FLOOPS, the fully-coupled model is still used as the basis for abstraction. In a decomposition that parallels SUPREM IV's data structures

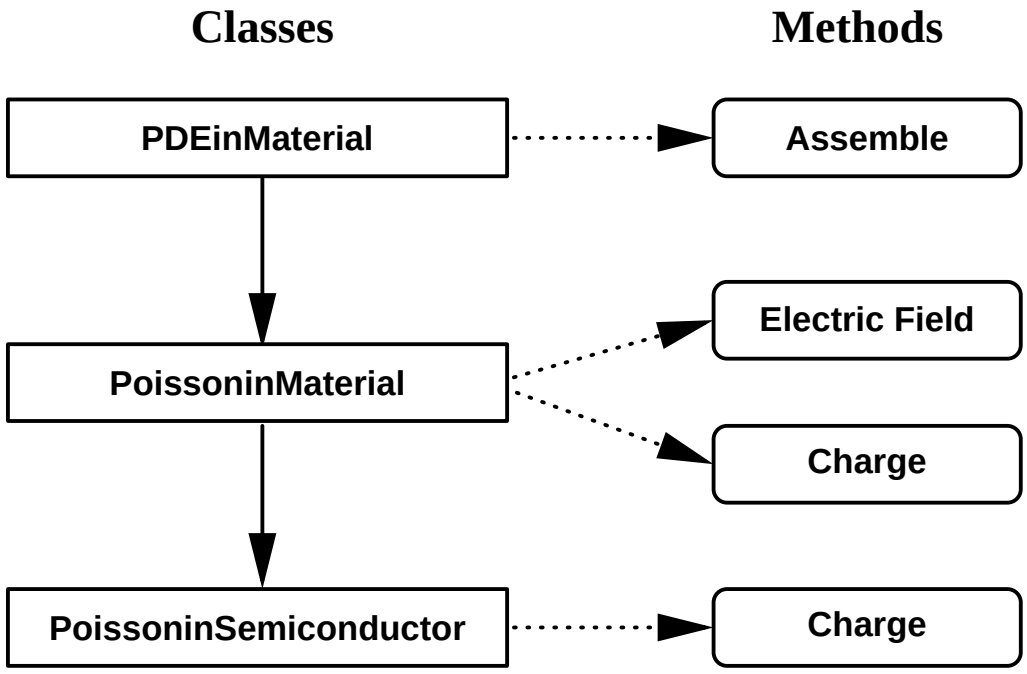

Figure 4.5: Class hierarchy for Possion's equation in FLOODS

and code organization, the SUPREM IV data structures used to store reusable quantities, such as the active portion of each impurity, diffusivity, and local carrier concentrations reappear in FLOOPS as classes that encapsulate and evaluate those quantities. Instead of the `switch/case` constructs used in SUPREM IV to handle material dependencies, FLOOPS defines a hierarchy of classes derived from the `PDEinMaterial` (`PDEinMaterial` $\longrightarrow$ `ConstantDiffusivity` $\longrightarrow$ `DopantinSemiconductor` $\longrightarrow$ `DopantwithDefect`) and instantiates instances of these classes to be used in the materials where they are appropriate.

While this decomposition does provide for limited sharing of some code and some class instance data, it does not provide a framework for extensibility to equation-based models that do not fit within the fully-coupled diffusion, such as treating either activation or dopant-defect pairing kinetically.

4.4.2 Inheritance in ALAMODE

In ALAMODE inheritance is not used to organize hierarchy in specific physical models. Instead, ALAMODE's design considered abstraction and hierarchy issues related to representation and discretization of equation-based models. The primary domain-specific use of inheritance is to provide a hierarchy of objects that perform interpolation elements in 1D, 2D, and 3D as well as the relationship with and mapping to a finite element isoparametric "parent" space. Other uses of inheritance are to provide type polymorphism in two areas: (1) the various forms of terms in an equation and (2) evaluation of unspecified scalar functions.

Hierarchies for element transforms

A finite element discretization typically makes use of transformations between a isoparametric parent space for elements and the computational space where instances of the elements exist as is shown in Figure 4.6. The geometric and field properties of this transformation are defined by a linear basis:

$$\vec{x}(\vec{\xi}) = \sum_{a=1}^{N_G} \vec{x}_a N_a(\vec{\xi}) \tag{4.3}$$

$$C(\vec{\xi}) = \sum_{b=1}^{N_F} C_b N_b(\vec{\xi}) \tag{4.4}$$

where $\vec{x}$ represents a coordinate in the computational space, $\vec{\xi}$ represents a coordinate in the isoparametric parent space, $\vec{x}_a$ are the ordered coordinates defining the vertices and other locations in a specific element, C_b are the field values at certain locations in a specific element, N_a are the interpolation functions for the linear basis for the geometric transform, N_b are the interpolation functions for the linear basis for the field transform, and $N_{G,F}$ are the number of knots in the linear basis for the geometric and field transforms, respectively. In a 1D Cartesian coordinate system, $\vec{x} = x$, and $\vec{\xi} = \xi$. In a 2D Cartesian coordinate system, $\vec{x} = [x \;\; y]'$, and $\vec{x} = [\xi \;\; \eta]'$. In a 3D Cartesian coordinate system, $\vec{x} = [x \;\; y \;\; z]'$, and $\vec{x} = [\xi \;\; \eta \;\; \zeta]'$.

ALAMODE only implements evaluation of transformed field quantities needed

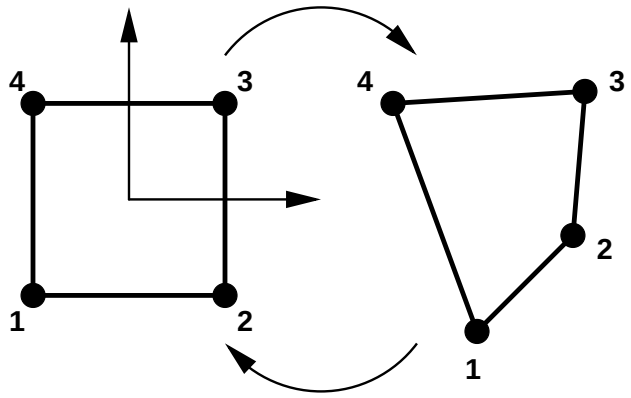

Figure 4.6: Mapping between an isoparametric parent space and computational space for a quadrilateral element.

for the term discretizations that are used in the equations related to semiconductor diffusion models. In addition to the evaluation of scalar field values (Equation 4.4), the flux discretization terms also need spatial gradients to be evaluated. With a 2D Cartesian transform, the spatial gradients are computed using:

$$\begin{aligned} \nabla_{\vec{x}} C &= \begin{bmatrix} \frac{\partial C}{\partial x} \\ \frac{\partial C}{\partial y} \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix}}_{\frac{\partial \vec{\xi}}{\partial \vec{x}}} \underbrace{\begin{bmatrix} \frac{\partial C}{\partial \xi} \\ \frac{\partial C}{\partial \eta} \end{bmatrix}}_{\nabla_{\vec{\xi}} C(\vec{\xi})} \end{aligned} \tag{4.5}$$

Extension to 3D is straightforward.

In addition to evaluation of the coordinate and file transforms and the spatial gradients, numerical quadrature is used to approximate integration of the finite element kernels. For example,

$$\int_{\Omega^e} f(\vec{x}) d\Omega \approx \sum_{l=1}^{N_{QP}} W_l f(\vec{x}(\vec{\xi_l})) j(\vec{\xi_l}) \tag{4.6}$$

where f is the integration kernel, N_{QP} is the number of quadrature points, W_l are

the quadrature weights, $\vec{\xi_l}$ are the quadrature locations in the isoparametric parent space, and j is the determinant of the geometric Jacobian:

$$j = \left| \frac{\partial \vec{x}}{\partial \vec{\xi}} \right| . \tag{4.7}$$

In ALAMODE, an object-oriented decomposition was applied to design a collection of cooperating classes which implement these evaluation requirements for a finite element discretization. The result is a hierarchy of element transform classes for both geometric and field transforms. This hierarchy is tightly related to classes that are used for storing geometric and field connectivity (`GeometricElement` and `FieldElement`). The latter two classes contain the methods used for evaluation of quantities needed for the finite element discretization. A number of other classes aid in the implementation of this hierarchy and its application to efficient finite element evaluation. These include:

- `ShapeFunction`: This class aids implementation of leaf element transforms of `GeometricElementTransform` and `FieldElementTransform` by providing an abstraction to implement shape functions and the required derivatives, taken with respect to the parent space, as a table of functions.

- `QuadraturePoint`: This is primarily a data class that contains a location in the parent space, $\vec{\xi}$. In addition to a `QuadraturePoint` class, there are also a number of `QP`*x*`D`*type* classes, where *x* is the dimensionality (0, 1, 2, or 3) and *type* specifies a coordinate convention for the parent domain, `Cartesian`, `Triangular`, `Prism`, or `Tetrahedral`. These classes ensure compatibility with specific instances of `ParentDomain`. For example, trying to use an instance of `QP2DCartesian` with the instance of `ParentDomain` associated with the 3D tetrahedral space would trigger an error.

- `QuadratureScheme`: This is a primarily data class that contains the weights, and quadrature points required for a quadrature scheme that will approximate an integral to a specific order of accuracy.

- `ParentDomain`: This class defines a legitimate subset of the parent space where element shape functions are defined. It also manages collections of quadrature schemes of varying accuracy that can be applied to integrate functions over elements compatible with the parent space. Each `ParentDomain` references all compatible leaf classes derived from `GeometricElementTransform` and `FieldElementTransform`, and directs pre-evaluation of shape functions for each transform at each quadrature point. This pre-evaluation provides for an inexpensive table lookup to be used in evaluation of Equations 4.3 and 4.4 instead reevaluating the shape functions every time they are needed.

The inheritance structure of the geometric element transforms is shown in Figure 4.7. `GeometricElementTransform` is a virtual base class containing declarations for all transform-related methods that may be called by an instance of `GeometricElement`. The abstract base class, `GeometricElementTransform`, provides *pure virtual*[1] methods to evaluate $\vec{x}(\vec{\xi_l})$ and $j(\vec{\xi_l})$ and a method to precompute shape functions at the quadrature points in a `ParentDomain`. The subclasses at the dimensionality level provide the method to evaluate $\vec{x}(\vec{\xi_l})$ and to initialize the tables of shape function values. The subclasses at the interior/boundary level provide the method to evaluate the geometric Jacobian matrix and its determinant ($j(\vec{\xi_l})$). For boundary element transforms, only the determinant is useful as a length, area, or volume multiplier for numerical integration. For interior element transforms, the geometric Jacobian matrix is used to transform gradients between the parent domain and computational space.

The leaf classes implement specific element transforms, providing the actual shape functions for the transform. Leaf classes also provide the only method to create instances of geometric elements based on the transform. Only leaf classes can be instantiated because C++ prohibits instantiation of abstract base classes, i.e. classes which either declare pure virtual methods or inherit unsatisfied pure virtual methods.

[1] In C++, a *pure virtual* method is a only a declaration of an interface. The method name, return type, and method arguments are provide, but no code will associated with the pure virtual method for class in which it is defined. Any code associated with a pure virtual method will be provided by a definition of the method in a derived class.

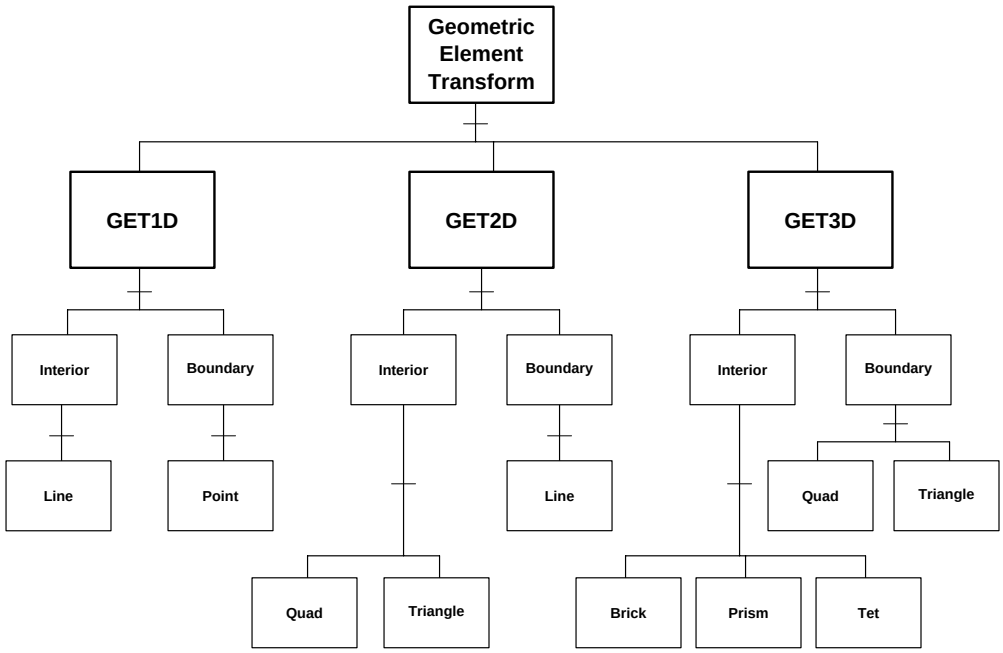

Figure 4.7: Inheritance hierarchy of geometric element transforms in ALAMODE.

The inheritance structure of `FieldElementTransform` (Figure 4.8) is nearly identical to that of `GeometricElementTransform`, but the capabilities provided differ. Like `GeometricElementTransform`, the `FieldElementTransform` is an abstract base class. It provides an interface to methods that:

- return the order of the interpolation basis
- return the order of the gradient of the interpolation basis
- evaluate a specific shape function or all of the shape functions at a specific quadrature point
- evaluate the value of the field at a quadrature point ($C(\vec{\xi}_l)$) or at a number of quadrature points
- evaluate the parent gradients of the field at a quadrature point ($\nabla_{\vec{\xi}} C(\vec{\xi})$) or at a number of quadrature points

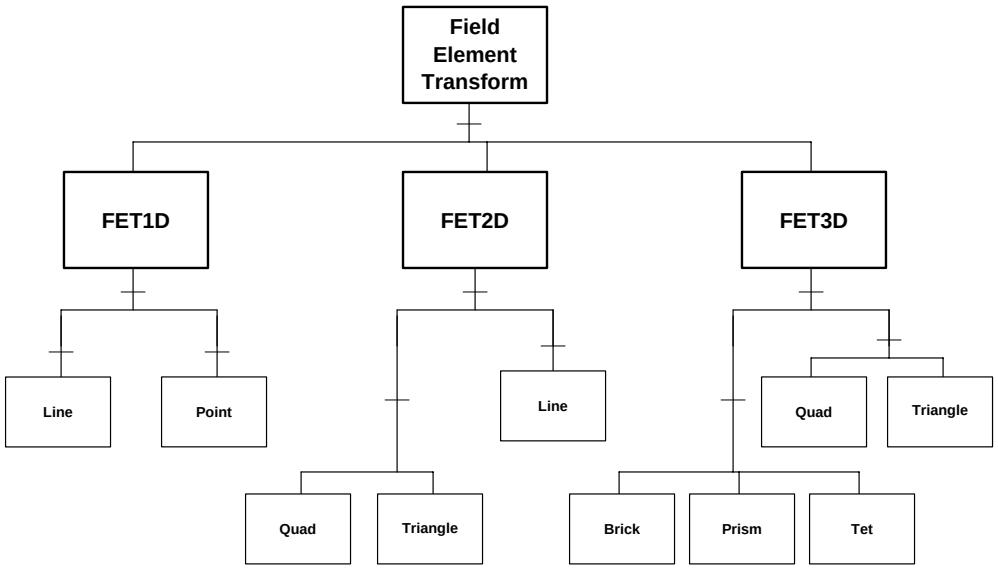

Figure 4.8: Inheritance hierarchy of field element transforms in ALAMODE.

- precompute shape functions at quadrature points in a `ParentDomain`

At the dimensionality level of the hierarchy, the evaluation methods are implemented as is the method to precompute shape functions at quadrature points. The leaf classes specify the order the interpolation basis, provide the actual shape functions for the transform, and provide a method to instantiate field elements based on the transform.

These hierarchies provide a framework in which to implement geometric and field transforms needed for finite element discretizations, assuming that the transforms are defined by a linear basis. This hierarchy provides a natural extensibility for each of the following:

- new element transforms, such as 2D triangles with a quadratic basis, can be added by creating a new class derived from the appropriate subclass of the field or element transform,

- additional quadrature schemes can be added in most cases by specifying the coordinates, multiplicity on the coordinates, and weights in a table that is used to create quadrature schemes for a given parent domain

- cylindrical or spherical coordinate systems can be added via additions to the quadrature point class structure and additional instances of `ParentDomain`

Although these hierarchies provide sufficient functionality to satisfy the finite element discretization needs for the terms in the models considered by ALAMODE, they are lacking in capabilities for terms needed for models in other finite element discretization domains such as structural mechanics and computational fluid dynamics. The main limitations that need to be addressed are support for vector fields and tensors and support for higher order derivatives of shape functions. Currently, only scalar fields are supported, and `ShapeFunction` only specifies first order differentiation with respect to each primary coordinate in the parent domain.

Polymorphism for Operators and Functions

The second significant use of inheritance in ALAMODE is to provide a common interface for terms in an equation and for the evaluation of scalar quantities. The common interface specifies the methods that need to be implemented by the derived classes which implement a specific discretization of a term or a specific functional form.

From the perspective of any discretized nonlinear PDE solver, a term in an equation will take as inputs the geometric and field properties of a mesh element and produce an element residual vector and tangent matrix. In ALAMODE, the interface for an `Operator` is:

```
class Operator {
public:
    virtual int isDependentOn( const FieldInRegionID& field ) = 0;

    int requestsNodalEvaluation();
    int requestsElementEvaluation();

    void attachForWeakEvaluation( HECID& hecID );
```

```
    void detachFromWeakEvaluation();

    virtual void addToWeakResidual(
        GeometricElementID& geID,
        double* residual, int rows ) = 0;
    virtual void addToWeakTangent(
        GeometricElementID& geID,
        const FieldInRegionID& wrtField,
        double** tangent, int rows, int cols ) = 0;

    void attachForNodalEvaluation();
    void detachFromNodalEvaluation();

    virtual const double* evaluateUnweightedNodalResidual();
    virtual void evaluateUnweightedNodalTangent(
        const double** nodalTangent,
        int nEQ,
        FieldInRegionID* unknownFields );

protected:
    virtual int weakOrderOfAccuracyNeeded( HECID& hecID ) = 0;
};
```

This interface provides for both element-based and nodal evaluation of the discretized weak form of a term in an equation. Derived classes of `Operator` provide the code to evaluate residual and tangent quantities. They also provide a method to determine if the term has an evaluation dependency on an instance of a field, which is essential for determining the structure of the tangent matrix.

In the process of evaluating the tangent or residual quantities, discretized terms need access to scalar values, gradient information, and derivatives of the quantities taken with respect to native fields. The `ScalarQuantity` abstract base class defines

the interface for evaluation of these quantities. Again, both element-based and nodal evaluation models are supported.

```
class ScalarQuantity {
public:
    // element evaluation
    virtual void eval( double* dest,
                       const GeometricElementID& geID ) = 0;

    virtual void evalDerivWrt( double* dest,
                               const FieldInRegionID& wrt,
                               const GeometricElementID& geID ) = 0;

    virtual void evalParentGradient( ParentGradient* dest,
                                     const GeometricElementID& geID ) = 0;

    // nodal evaluation
    virtual void eval( double* dest, int n ) = 0;
    virtual void evalDerivWrt( double* dest, int n,
                               const FieldInRegionID& wrtField ) = 0;

    virtual int dependsOn( const FieldInRegionID& field ) const = 0;

    virtual int orderOf( HECID& hecID ) const = 0;
    virtual int orderOfGradient( HECID& hecID ) const = 0;

    int operator==( const Evaluable& evalable ) const = 0;
};
```

In addition to the evaluation routines, the abstract class definition provides a method to detect dependencies on native fields and methods which approximate the

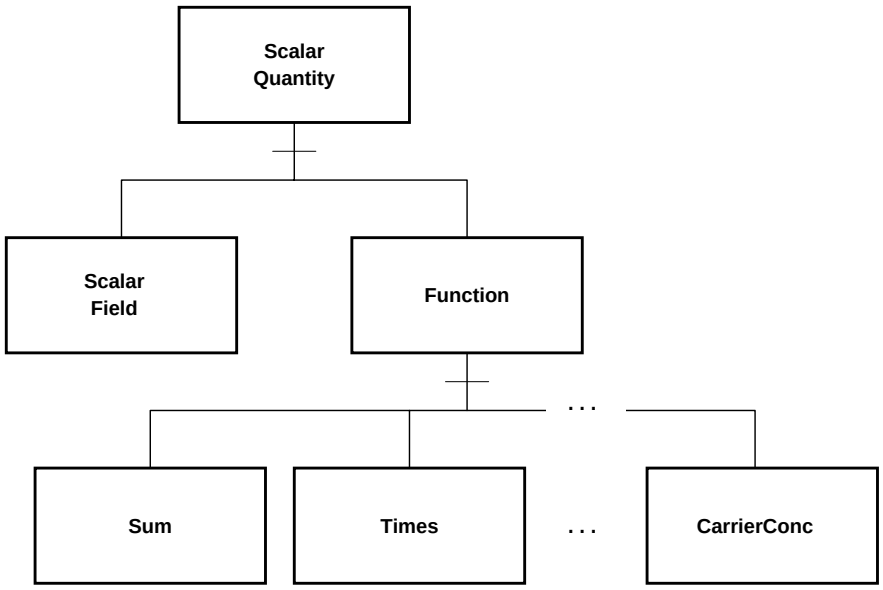

Figure 4.9: Inheritance hierarchy for evaluation of scalar quantities.

interpolation order of the scalar quantity. These dependency methods are used in definition of the matrix structure and for automatic differentiation (see Section 5.1.3). The interpolation order estimates are used to select an appropriate numerical quadrature scheme for operators that implement a consistent FEM discretization.

The inheritance hierarchy of the classes derived from `ScalarQuantity` is shown in Figure 4.9. `ScalarField` implements evaluation and caching of these quantities for native fields. `Function` provides an intermediate abstract class that implements evaluation caching for functional forms, but leaves the responsibility of the actual evaluation of these forms up to the leaf classes. Caching of evaluated quantities is discussed in Section 5.1.3.

4.5 Summary

This chapter has examined practical software design issues which are generally not discussed in depth by proponents of specific design methodologies. To produce a design with enough flexibility to be easily extended, one must avoid basing the design

on abstractions and assumptions that are likely to be invalidated during the lifetime of the software. In the design of TCAD software, there is tendency to develop implementation abstractions based on specific physical models, but this should be avoided when the specific physical models are known to have a limited lifetime.

Increasing complexity of CAD simulation software has motivated a second design issue, the need to design for integration. Software packages are no longer developed as a single standalone entity. They must integrate with external packages, and often several different external packages that implement similar functionality, such as the solution of systems of linear equations. The advice developed here is that a generic interface should be developed, and that all external packages integrated through that interface should use only the methods provided by the generic interface. Two examples of generic interfaces in ALAMODE were presented, one which communicates with linear solvers and one that provides for interchange of mesh and field data.

Finally, a good deal of general CAD software development has moved to object-oriented programming techniques and an implementation in an object-oriented language such as C++. Unfortunately, this alone does not prevent implementations that may suffer from the same extensibility problems caused by basing design abstractions on assumptions with a limited lifetime. Although object-oriented inheritance seems to provide a very natural mechanism for extensibility, the extensibility inheritance provides is still limited by the abstractions used to develop the hierarchy. One must be very careful about the implications and limitations that may result by developing an inheritance hierarchy to model the relationships among a group of objects.

Chapter 5

Implementation

The previous chapter discussed design issues, primarily motivated by extensibility, integration, and maintenance perspectives, and developed a hierarchical decomposition of the primary objects that implement a dial-an-operator discretization, namely discretized terms in an equation and objects that provide for evaluation of arbitrary expressions. The implementation of these objects is now considered, with the primary emphasis on efficiency.

5.1 Expression evaluation and differentiation

A dial-an-operator simulator uses an internal data structure to describe arbitrary coupled systems of partial differential equations appropriate for diffusion models, but it also needs to be able to efficiently compute and solve the discretized form of these equations on a mesh. The information model and data structures for model representation have been considered in Chapter 2. This section concentrates on the techniques for automatically computing derivatives from the equation representation and for efficiently evaluating the discretized equations on a variety of mesh elements. It is assumed that all PDE systems in various regions are globally coupled and that a global Newton approach will be used to solve the nonlinear system of algebraic equations that result from the spatial and temporal discretizations. Each region is assembled independently into a globally coupled residual vector and tangent matrix.

For the purpose of discussion of element-based assembly within a region, consider the following system

$$\begin{aligned} E_1 : \frac{\partial C_1}{\partial t} &= T_{1,1}(C_1, C_2, C_3) + T_{1,2}(C_1, C_2) + T_{1,3}(C_1, C_3) \\ E_2 : \frac{\partial C_2}{\partial t} &= T_{2,1}(C_1, C_2, C_3) + T_{2,2}(C_1, C_2) + T_{2,3}(C_2) \\ E_3 : \frac{\partial C_3}{\partial t} &= T_{3,1}(C_3) + T_{3,2}(C_1, C_2) \end{aligned} \tag{5.1}$$

where $T_{a,b}$ represents a term that depends on unknown fields, C_a. As with most finite element programs, element residuals and tangent matrices are linearly independent across elements, allowing the globally coupled system to be computed as an assembly (or sum) of those element-based quantities. In a typical finite element program (e.g. FEAP [56]), the element routines compute the element residual vector and tangent matrix for the complete locally coupled system of equations. Thus, given a specific element (e) in the mesh, the element evaluation routine would produce

$$\mathbf{r}^e = \left\{ \begin{array}{c} \mathbf{r}^e(E_1) \\ \mathbf{r}^e(E_2) \\ \mathbf{r}^e(E_3) \end{array} \right\} \tag{5.2}$$

and

$$\mathbf{t}^e = \left\{ \begin{array}{ccc} \mathbf{t}^e_{1,1} & \mathbf{t}^e_{1,2} & \mathbf{t}^e_{1,3} \\ \mathbf{t}^e_{2,1} & \mathbf{t}^e_{2,2} & \mathbf{t}^e_{2,3} \\ \mathbf{t}^e_{3,1} & \mathbf{t}^e_{3,2} & \mathbf{t}^e_{3,3} \end{array} \right\} \tag{5.3}$$

where $\mathbf{t}^e_{a,b}$ is the discretized tangent obtained by differentiating the residual for equation E_a with respect to the interpolated field variable C_b.

Because the operators (i.e. terms in an equation) are also linearly independent, the assembly operation can be extended to each term in the equation. Thus, instead of evaluating a complete system residual and tangent for an element, each operator only needs to evaluate its own contribution. This is the approach used in ALAMODE, primarily because it maximizes reuse of code. The diffusion/flux operator need only

have one implementation that can be reused whenever needed.

For the system described above,

- $T_{1,1}$ contributes to $\mathbf{r}^e(E_1)$, $\mathbf{t}^e_{1,1}$, $\mathbf{t}^e_{1,2}$, and $\mathbf{t}^e_{1,3}$
- $T_{1,2}$ contributes to $\mathbf{r}^e(E_1)$, $\mathbf{t}^e_{1,1}$, and $\mathbf{t}^e_{1,2}$
- $T_{1,3}$ contributes to $\mathbf{r}^e(E_1)$, $\mathbf{t}^e_{1,1}$, and $\mathbf{t}^e_{1,3}$
- $T_{2,1}$ contributes to $\mathbf{r}^e(E_2)$, $\mathbf{t}^e_{2,1}$, $\mathbf{t}^e_{2,2}$, and $\mathbf{t}^e_{2,3}$
- $T_{2,2}$ contributes to $\mathbf{r}^e(E_2)$, $\mathbf{t}^e_{2,1}$, and $\mathbf{t}^e_{2,2}$
- $T_{3,3}$ contributes to $\mathbf{r}^e(E_2)$ and $\mathbf{t}^e_{2,1}$
- $T_{3,1}$ contributes to $\mathbf{r}^e(E_3)$ and $\mathbf{t}^e_{3,3}$ $\mathbf{t}^e_{3,3}$
- $T_{3,2}$ contributes to $\mathbf{r}^e(E_3)$, $\mathbf{t}^e_{3,1}$, and $\mathbf{t}^e_{3,2}$

5.1.1 Evaluation of operator quantities

During the assembly of the global residual vector and tangent matrix, instances of derived classes of `Operator` will be called on to evaluate their contributions to the element residual vector and element tangent matrix. In the interface defined by `Operator`, the methods which perform this computation are declared as follows:

```
void addToWeakResidual(
    GeometricElementID& geID,
    ElementResidual residual, int rows );
void addToWeakTangent(
    GeometricElementID& geID,
    const FieldInRegionID& wrtField,
    ElementTangent tangent, int rows, int cols );
```

The location of the residual vector that is passed refers to the subset of the element residual vector associated with the equation for that term. Each element tangent matrix that is passed refers to the subset of the region's element tangent associated with the equation for that term and the unknown field against which the differentiation is being performed. For example, in the evaluation of the residual for $T_{2,2}$, the location of $\mathbf{r}^e(E_2)$ would be passed. In the evaluation of $\frac{\partial T_{2,2}}{\partial C_1}$, only the locations of $\mathbf{t}^e_{2,1}$ would be passed.

5.1.2 Representation of scalar expressions

The operator classes exclusively utilize objects of type `ScalarQuantity` instead of hard-coding any constitutive models. This provides the ability to reuse the operator class for a variety of user-defined constitutive models. For example, the diffusive flux operator ($\nabla \cdot D \nabla C$) treats both D and C as scalar quantities. Thus, scalar quantities need be able to represent arbitrary scalar expressions.

A tree data structure is used to represent the expression. Each node in the tree is an instance of a derived class of `ScalarQuantity`. The derived classes under `Function` provide a full set of arithmetic operators (addition, subtraction, multiplication, and division) as well as many auxiliary functions needed in the domain of diffusion modeling. Leaf nodes must either be `ScalarField` or analytic function that is independent of interpolated field values. An example of an expression tree is shown in Figure 5.1.

5.1.3 Evaluation of expressions

Evaluation of expressions used by the spatially discretized operators could be performed as most interpreted languages evaluate expressions, via single quantity *on-demand* implementation. In an on-demand implementation, the value for each expression is evaluated just before it will be used. For example, the expression

$$\exp(B * \log(C)) * A + D$$

would be evaluated by a machine using the following sequential steps:

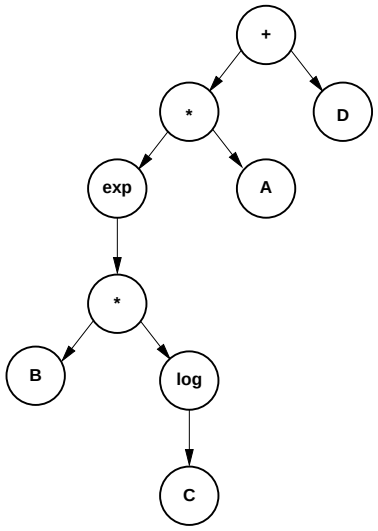

Figure 5.1: Expression tree for $\exp(B * \log(C)) * A + D$.

1. fetch value of C
2. call log
3. fetch value of B
4. multiply value of B by result of $\log(C)$
5. call exp
6. fetch value of A
7. multiply by the result of $\exp(B * \log(C))$
8. fetch value of D
9. add to the result of $\exp(B * \log(C)) * A$

To apply on-demand evaluation to expressions represented as a tree, a depth-first traversal is performed, and the arithmetic operation or function is applied to the results of each subtree after the subtree is fully traversed. Demand evaluation is not difficult to implement, but it is generally not optimally efficient when compared

to compiled code implementing the same expression. Inefficiencies come primarily from three sources, tree traversal and interpretation overhead, removal of instruction-level parallelism and the ability of a compiler to produce an optimal sequence of instructions, and duplicated evaluation of identical subtrees.

Traversal and interpretation overhead

The first source of inefficiency comes from the need to access additional memory locations unrelated to the actual evaluation of the expression while traversing each expression node, to identify the node, and to jump to instructions that perform the actual evaluation. The actual relative performance impact will depend on the quality of the implementation of the expression evaluator, the expression being evaluated, and the characteristics of CPU where the code is run. To quantify the performance impact, consider

$$T_I = M + C + L + S + T_O + J \tag{5.4}$$

where T_I is the average time to evaluate the expression tree at each node, M is the average time associated with the additional memory accesses to traverse the tree, C is the average time associated with the compare and jump to the instructions that evaluates the node, L and S are the average times to load and store intermediate results[1], T_O is the average time to execute the instructions that evaluate the node, and J is the cost to jump back to the start of the interpretation loop.

Traversal and interpretation overhead will be a larger portion of evaluation time for expression trees consisting of inexpensive arithmetic operations than for trees with more expensive operations such as e^x. A worst-case estimate can be obtained assuming that each operation takes a single execution unit ($T_O = 1$). For binary nodes, there would be one downward and one upward memory access required for traversal of each branch for a total of four traversal-related memory accesses. Ignoring

[1]Additional load and store penalties are assessed for interpreted evaluation but not compiled evaluation because it is assumed that a compiler will generate code to store intermediate results in registers. On-demand interpreted evaluation cannot use registers because the number of intermediate results that need to be saved is not known. Thus, memory must be used.

address calculations, these are assigned a cost of $M = 4$. Compare and jump is assigned a cost of $C = 2$. Load and store are each assigned costs of $L = 1$ and $S = 1$ because the intermediate result obtained from evaluation of one branch could be stored in a register. The cost to jump back is assigned $J = 1$. With these parameters, on-demand interpreted evaluation would, in the worst case, run 10 times slower than compiled code due only to the overhead of interpretation. More expensive operations, such as e^x would have a smaller portion of total evaluation time spent in interpretation overhead.

This performance impact of traversal cannot be eliminated, but it can be amortized by operating on blocks of values instead of individual values. The main drawback to block evaluation is that the implementation becomes more complicated. Each expression node may have to allocate and deallocate blocks of memory for temporary storage of vectors of values. Load and store penalties may also be increased because of the use of arrays to store intermediate results, but on many machines, accesses to sequential memory locations can be overlapped with execution, yielding no penalty as long as sufficient memory ports and bandwidth are available.

Instruction-level parallelism

Obtaining optimal execution efficiency on a modern computer also requires writing code that allows the best use of the processor's pipeline [16]. Modern RISC processors and microarchitectures do not need to wait until an instruction completes before issuing the next instruction. Instead, several instructions can be in various stages of execution within an execution pipeline. If a compiler has all information about the expression to be evaluated, it can reorder the instructions that evaluate the expression to get the optimal execution efficiency from the processor's pipeline. This changes the other costs relative to T_O, especially in a comparison of the cost of single quantity interpreted evaluation versus optimal block evaluation of the same expression in compiled code. Again, the actual relative performance impact will depend on several factors, such as quality of code generated for an expression by a compiler, the

expression being evaluated, and the characteristics of the CPU. Define

$$E = \frac{o}{cm} \tag{5.5}$$

where E is the effectiveness of evaluating the expression, o is the number of operations in the expression, c is the number of clock cycles needed to evaluate the expression, and m is the maximum number or operations that can be issued or retired in each cycle. Expressions that contain sufficient instruction- and loop-level parallelism map well onto most pipelined architectures and can approach an effectiveness of $E = 1$, keeping all functional units as busy as possible and avoiding delays caused by dependency-related pipeline stalls or improperly predicted jumps.

Code which interprets single quantity expressions has significantly less intrinsic parallelism and must be executed sequentially. Furthermore, successive instructions often depend on the result of the previous instruction. This causes the pipeline to stall until the result is available, adding more cycles to the execution time. Interpretors also jump out of sequence more frequently, and the targets of the jump are dependent on data, such as the type of the operand node. If the target of the jump is not predicted correctly, the processor must discard the incorrect target's instructions in the fetch and decode stages of the pipeline and fetch the instructions at the correct target address. The processor features that provide improved performance for code containing significant instruction- and loop-level parallelism do not significantly improve the performance of code implementing interpretation.

To provide some actual numbers for the aggregate penalty for interpretation, consider a comparison of the LINPACK [15] benchmark in compiled and interpreted environments on the same machine. For the compiled case, the unmodified double precision FORTRAN LINPACK benchmark was applied to a matrix of size 100×100. For the interpreted environment, a Java version the double precision LINPACK benchmark was applied to a matrix of the same size using a Java Virtual Machine that performed strict bytecode interpretation. On a Sun UltraSPARC running at 167 MHz, the compiled benchmark achieved 70 million floating point operations per second (MFlop/s) while the interpreted version only achieved 0.44 MFlop/s. This

penalty is clearly unacceptable for large-scale computation, and it is questionable if it would be accepted even in a prototyping environment.

Evaluating expressions nodes on blocks of data instead of individual values provides a better opportunity for the compiler to produce an efficient sequence of instructions for evaluating the reduction or transformation represented by that node. Even though block evaluation allows a compiler to discover more loop-level and instruction-level parallelism, some potential optimizations may still be missed. For example, many architectures, such as the IBM POWER CPU and the Intel i860, support chaining of a floating point multiply operation with a floating point addition operation, effectively issuing two floating point operations per cycle. If the expression tree does not support architecture-specific features and modify the expression tree to capitalize on these features, optimal efficiency will not be achieved. The worst-case performance penalty for not considering these architecture-specific features is related to the best-case speedup obtained for code that utilizes the feature as compared to code that implements the same functionality without utilizing the feature. For the multiply-and-add feature, the worst-case performance penalty would be a factor of two. Penalties on this order are acceptable for prototyping environments.

The actual overall speedup (S) of such optimizations is limited by Amdahl's Law,

$$s = \frac{t_{\text{base}}}{t_{\text{opt}}} = \frac{1}{1 - f + \frac{f}{s}} \tag{5.6}$$

where f is the fraction of the base run time that can be enhanced by a particular feature, and s is the speedup of enhancement. Unless f is close to 1, the overall speedup is seldom close to the speedup of the enhancement. Missing out on such optimizations may even be acceptable for large-scale computation.

Duplicated expressions

The third major source of inefficiency, duplicated expressions, can only be addressed by detecting the duplicated expressions and ensuring that any duplicated subtrees will be evaluated only once. Unlike the other sources of inefficiency, the performance

impact due to duplicated evaluation of these expression depends only on the expressions used in the model. Eliminating duplicated expressions is especially important for semiconductor diffusion models because they are very common in the equations that comprise these models. In reduced equation models, such as the Fermi or fully-coupled formulations, diffusivities and potential are dependent on the carrier concentration. In fully kinetic models, pairing and dissociation reactions appear as repeated expressions in multiple equations. Because each individual operator in a system of equations could utilize expressions that contain duplicated subtrees, the detection should be performed at the *system in region* level.

The above representation of expressions, which uses disjoint tree structures that are only rooted at operators, would need to be altered to allow for sharing of nodes at the head of duplicated subtrees. Evaluation also cannot use a purely root-to-leaf (demand) traversal because any shared expressions should only be evaluated once, and the results should be saved so that they can be reused by other expression trees. The approach taken in ALAMODE is to abandon traversal of the operator-rooted trees to provide a sequential order for expression evaluation. Instead, the operator-rooted expression trees are preprocessed and flattened into an *evaluation dependency* ordered list of non-duplicated evaluation nodes. *Evaluation dependency* order means that each node must appear in the list after all nodes in the subtree below that node. The algorithm is illustrated by the pseudo-code in Figure 5.2 and Figure 5.3.

Note that ADD-IF-UNIQUE, recursively descends down to leaf nodes before attempting to add any nodes to the list and can only add the current node after all subtrees have been traversed. This guarantees that nodes are added in evaluation dependency order.

Also note that if an equivalent expression subtree has already been added to the list, the node will obtain a reference to that subtree as its cache node (SET-CACHE-NODE. When requested for its values, a tree node that is not in the evaluation list will obtain its values from its master cache node.

The remaining primitive operation, FIND-EQUIVALENT-NODE, simply searches the list of nodes, looking for one that is algebraically equivalent to the specified node, as is shown in Figure 5.4.

```
BUILD-EXPRESSION-LIST(S)
Input: A system in region, S
Output: A list of unique, evaluation-dependency-order sorted expression nodes.
(1)   E ← ∅
(2)   ; loop over equations in the system
(3)   foreach e ∈ S
(4)      ; loop over operators in the equation
(5)      foreach o ∈ e
(6)         T ← EXPRESSIONS-USED-BY-OPERATOR(o)
(7)         ; loop over expressions used by the operator
(8)         foreach t ∈ T
(9)            ADD-IF-UNIQUE(t, E)
(10)  return E
```

Figure 5.2: An algorithm to traverse all expression trees used by the operators in a *System in Region* in order to build a dependency-ordered expression list.

```
ADD-IF-UNIQUE(t, E)
Input: A node in an expression tree, t
Output: The list of expression nodes, E, modified in place.
(1)   T ← SUBTREES-OF-NODE(t)
(2)   foreach e ∈ T
(3)      ADD-IF-UNIQUE(e, E)
(4)   N ← FIND-EQUIVALENT-NODE(E,t)
(5)   if N = ∅
(6)      APPEND(E, t)
(7)   SET-CACHE-NODE(t,N)
```

Figure 5.3: An algorithm to recursively descends subexpressions, adding expression nodes to a list if an equivalent expression is not already in the list.

```
FIND-EQUIVALENT-NODE(E, t)
Input: A list of nodes, E, and a specific node in an expression tree.
Output: Returns the equivalent node in E is found or ∅.
(1)   foreach e ∈ E
(2)     if ARE-EQUIVALENT(e, t)
(3)       return e
(4)     else
(5)       return ∅
```

Figure 5.4: An algorithm to search a list of expression nodes for an equivalent expression.

Each `ScalarQuantity` implements a test, ARE-EQUIVALENT, appropriate the operation that it implements. For example, the addition operator considers two addition nodes, $(a_1 + b_1)$ and $(a_2 + b_2)$, to be equivalent if ARE-EQUIVALENT(a_1, a_2) and ARE-EQUIVALENT(b_1, b_2) or if ARE-EQUIVALENT(a_1, b_2) and ARE-EQUIVALENT(b_1, a_2).

Checks for equivalence are performed without modification of the expression trees. A drawback is that not all equivalent trees can be discovered. For example, $(a+b)+c$ will not be detected as being equivalent to $a + (b + c)$. In practice, this is not a significant drawback as model developers usually write equivalent expressions in the a detectable form.

Automatic differentiation

Solution of a system of nonlinear equations using a Newton approach requires both the residual vector and an estimate of derivative of this vector taken with respect to nodal solution variables, i.e. a tangent matrix. As was discussed above, the residual vector is fairly easy to evaluate in an interpreted environment, given the list of terms and a set of reusable functions. Computation of the tangent matrix, however, requires computation of derivatives taken with respect to solution variables. An additional complication is that it is not know which fields will be solution variables until the model is provided as input to the simulator. Thus, differentiation must also be part of the expression interpretation.

The derivative quantities could be estimated using numerical differentiation, as is

done in [37]. This approach can be significantly more expensive than computation of the tangent matrix using a hand-coded analytic routine because it requires multiple evaluations of the residual vector with different perturbations of the nodal solution values. The accuracy of the estimate is subject to a *stepsize dilemma* [35]. Smaller perturbations decrease the truncation error, which is desirable, but they also increase the roundoff error. The optimal size of the perturbation is a heuristic parameter that can be determined only when the expression being differentiated is known. Thus, for both efficiency and accuracy reasons it is usually desirable to compute the derivative using an analytic formula.

The techniques for automatically computing the analytic derivative given an expression are called *automatic differentiation* [46]. Although there are several different approaches to computing the derivative, all rely on the systematic application of the chain rule to produce the analytic derivative value. Two of the general approaches that are in wide use include *source-to-source transformation* and *operator overloading* [14].

In source-to-source transformation, an external tool, such as ADIC [5], parses specially annotated program source code files to transform them into source code files that contain expressions to evaluate the derivatives. The special annotations in the input source code files provide auxiliary information, such as specifying the dependent and independent variables. The main drawbacks of this approach are: 1) that dependent and independent variables must be identified at the time of the source-to-source transformation and 2) the transformed source code must be compiled and linked with an application before it can be used.

The second approach only works with languages, such as C++ and FORTRAN 90, that support operator overloading. In this approach, the operator overloading facilities are used to replace native arithmetic operators and functions with routines that propagate derivative information in addition to the value of an expression. As an example of an overly simplified implementation in C++, consider the following example:

```
class autodiff {
public:
 ...
    friend autodiff operator * (autodiff&, autodiff&);
private:
    double f;
    double df;
};
autodiff  autodiff :: operator * (autodiff& a, autodiff& b)
{
    autodiff c;
    c.f = a.f * b.f;
    c.df = a.f * b.df + a.df * b.f;
}
```

Although this simplified fragment only shows differentiation with respect to a single independent variable, a more sophisticated implementation could provide for differentiation with respect to a set of independent variables that are not specified until run time. The main drawback of operator overloading is in achieving an efficient implementation. Context information, which is needed for optimizations such as common subexpression removal, is not present to the implementor of the overloaded operators. There may also be additional overhead associated with overloading, such as the need to call functions instead of generating inline code and the lack of control over the generation and destruction of temporary objects.

Both of the above general approaches to automatic differentiation are targeted at an implementation where the expressions and derivatives are coded in a programming language such as FORTRAN, C, or C++. In ALAMODE, the expressions in the model are input by a model developer and stored in an expression tree with the intent that they will be evaluated using optimized interpreted techniques. Although code could have been generated from the expression tree, then preprocessed, compiled,

and linked to create a new simulator, this approach was not attractive for several reasons. Portability was a concern, both in the availability of compilers and other preprocessing tools and in the mechanisms to invoke those external programs.

A secondary concern was that the time required to compile and link a new simulator would be greater than the time to simulate the model. This is particularly important to model developers because they may be changing the equations and terms in ways that would require a new compilation. It has been reported [34] that Taurus, a commercial "input your own equations" TCAD simulation environment that uses the code generation approach, can take a substantial period of time (i.e. 10 minutes) to generate a new simulator from an input specification.

In ALAMODE, differentiation is built into the `ScalarQuantity` class and handled as part of interpreted evaluation. Each `ScalarQuantity` subclass implements methods to compute the derivative with respect to a specified field:

```
virtual int dependsOn( const FieldInRegionID& field ) const = 0;

// element−wise
virtual void evalDerivWrt( double* dest,
                           const FieldInRegionID& wrt,
                           const GeometricElementID& geID ) = 0;

// nodally
virtual void evalDerivWrt( double* dest, int n,
                           const FieldInRegionID& wrtField ) = 0;
```

If the subclass is not the subclass of `ScalarQuantity` that implements access to the field values, the `evalDerivWrt` method uses the chain rule to propagate its contribution, based on the values and derivatives of the `ScalarQuantity` that the instance uses. For example, the implementation of the nodal `evalDerivWrt` for the $\exp(x)$ function is:

```
void ExpFunction::evalDerivWrt(
    double* dest,
    int n,
    const FieldInRegionID& wrtField )
{
    int iN;
    const double* funcValues;
    const double* funcDeriv;

    if ( funcTag_->isDependentOn(wrtField) ) {

        funcValues = funcTag_->getCachedNodalValues();
        funcDeriv = funcTag_->getCachedNodalDerivWrt(wrtField);

        for( iN = 0;  iN < n;  iN++ ) {
            dest[iN] = exp( funcValues[iN] ) * funcDeriv[iN];
        }
    }
}
```

This code fragment also partially exposes some of the implementation of the management of the shared evaluation cache. Instead of directly calling the methods which actually perform the computationally expensive evaluation, the evaluation routine uses a tag to access precomputed and cached values. The constructor for the subclass registers all instances of scalar quantities that are used by the function to the shared evaluation cache. This list of registered scalar quantities provides the data needed for the SUBTREES-OF-NODE(t) primitive used in the algorithms in section 5.1.3.

The constructor also informs the shared evaluation cache what values to precompute for different uses of the scalar quantity. For the $\exp(x)$ function, only the value of the quantity needs to be computed and available in the cache when an instance of

`ExpFunction` is used in a residual calculation, but both the value and derivative need to be computed and cached if the tangent matrix is being generated. The constructor for `ExpFunction` is:

```
ExpFunction::ExpFunction( ScalarQuantityID func )
{
    // Save a copy
    func_ = func;

    // Register all quantities used by this function
    funcTag_ = addScalarQuantity( func );

    // Quantities needed for eval
    funcTag_->addToValueFlags( CACHE_VALUES );

    // Quantities needed for evalDeriv
    funcTag_->addToDerivFlags( CACHE_VALUES );
    funcTag_->addToDerivFlags( CACHE_DERIV );

    // Quantities needed for evalParentGradient
    funcTag_->addToGradientFlags( CACHE_VALUES );
    funcTag_->addToGradientFlags( CACHE_PARENT_GRADIENT );
}
```

Note that it is not necessary to precompute and cache the values, derivatives, and/or gradients (VDG) that will be needed by an operator during a residual or tangent computation. Instead, a lazy approach could have been used to achieve the same goal, only computing VDG that are required for use by a particular operator. In the lazy approach, if the shared evaluation cache is requested to provide a VDG, and that VDG has not yet been computed and cached, then the evaluation cache will call upon the `ScalarQuantity` associated with cache tag to evaluate the requested

VDG. Doing so will cause further propagation of evaluation requests down the expression tree through intermediate cache tags until all subtrees have been evaluated and cached. Although similar to demand evaluation, the recursion down the tree will terminate as soon as the cached VDG quantity is found. Because each tag in the evaluation cache accesses the VDG quantities from its master cache node (i.e. the node at the head a possibly duplicated subtree) duplicated subexpressions are only traversed and evaluated once.

5.2 Further optimizations for efficiency

The shared cache implementation of interpreted expression evaluation provides operators with an efficient mechanism to compute the values, derivatives, and spatial gradients needed for discretization of the operator. However, this alone does not provide near optimal run time performance for evaluating discretized models. The element-by-element evaluation paradigm, which is a common implementation in finite element codes, is inefficient compared to nodal evaluation for evaluating nodally lumped reactive terms. To obtain the best performance from a sparse linear solver, the code should avoid storing any structural zeros. Each of these optimizations is discussed in more detail below.

5.2.1 Nodal evaluation

If a reactive term is nodally lumped, then the residual at the node only depends on values at the node. If the local degrees of freedom on an element are grouped by field, then the element tangent matrix contributions from nodally lumped terms would consist of blocks of diagonal submatrices. For example, consider the the system of equations shown in Equation 5.1. If the terms $T_{1,2}$, $T_{2,2}$, $T_{2,3}$, and $T_{3,1}$ are reactive terms that are treated nodally, then the structure of the element tangent matrix produced only by those terms for a triangular element with the local degree of freedom numbers shown in Figure 5.5 would be:

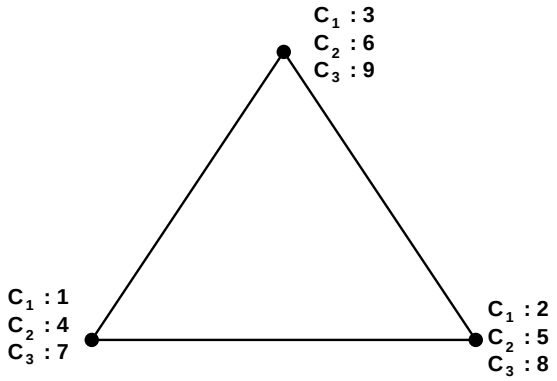

Figure 5.5: One possible convention for local degree of freedom numbers for multiple fields on an element.

$$
\begin{array}{c}
 \\
E_1: \\
 \\
 \\
E_2: \\
 \\
 \\
E_3: \\
 \\

\end{array}
\overbrace{\left\{\begin{array}{c}
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \\
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \\
\left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right]
\end{array}\right.}^{C_1}
\overbrace{\left\{\begin{array}{c}
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \\
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \\
\left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right]
\end{array}\right.}^{C_2}
\overbrace{\left\{\begin{array}{c}
\left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right] \\
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \\
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]
\end{array}\right.}^{C_3}
\tag{5.7}
$$

Even if the evaluation of a nodally lumped term were optimized to only compute the non-zero diagonal entries, an element-by-element assembly would still duplicate evaluation of some quantities at any node shared by multiple elements. Computing the global residual vector and tangent matrix contributions at node `5` of the mesh shown in Figure 5.6 using an element-by-element approach would require evaluating elements `b`, `c`, `d`, `e`, `f`, and `g`. Computing the same quantities from a nodal perspective, only requires multiplying the nodal values and derivatives by a measure of the area

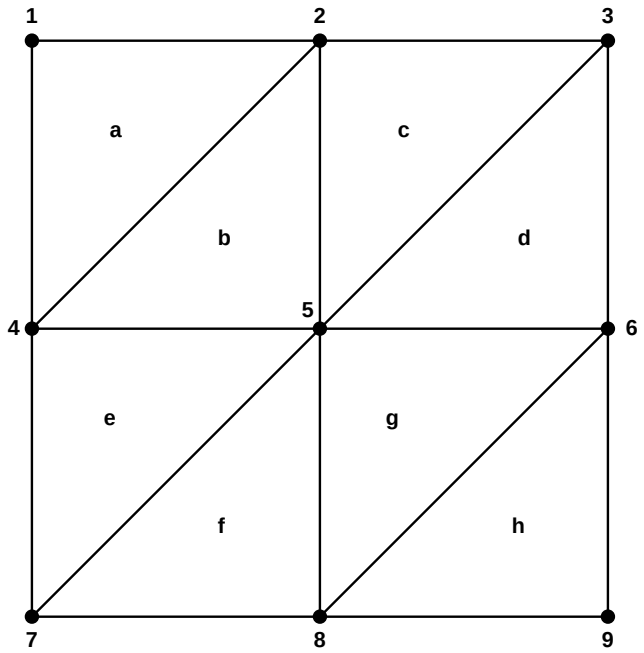

Figure 5.6: A sample mesh.

surrounding node 5. By operating on blocks of nodes, this operation can be very efficient.

Another inefficiency that was addressed with the introduction of nodal evaluation was duplication of the evaluation of nodal quantities for the geometric diffusion term. In a strict element-by-element paradigm, the discretized term would evaluate quantities at the quadrature points within an element, and the general case would not detect whether certain evaluation locations, such as the element nodes, were shared among elements. For the geometric diffusion discretization and the exponentially weighted diffusion discretization, the quadrature points are at the nodes of the element. These operators are treated nodally for expression evaluation, but evaluation of the spatial discretization is handled element-by-element. A gather operation collects the data needed for the local view of the expression values and derivatives from the nodal portion of the shared evaluation cache.

5.2.2 Optimal sparsity in the tangent matrix

Preconditioned sparse iterative techniques are used to solve the linearized system of equations ($\mathbf{T}\Delta\mathbf{C} = -\mathbf{R}$). The computational cost for obtaining a solution to this linear system is determined both by the cost of computing and applying the preconditioner and the cost of applying the iterative technique. In applying the iterative technique, the primary operation is a matrix-vector multiply, and the cost of this operation is roughly proportional to the number of structural non-zeros in the sparse matrix storage. There is more variation in the cost of the preconditioners, which may vary from having a fixed cost to a cost proportional to number of equations to a cost that depends superlinearly on the number of structural non-zeros. The most commonly used preconditioner in ALAMODE, an incomplete lower/upper (ILU) factorization has a cost that is at least linear in the number of structural non-zeros. Thus, to obtain the fastest linear solution times, one should avoid storing any structural zeros in the global tangent matrix. Structural matrix entries for the global tangent (T_{AB}) should only be allocated if the residual for the equation at global degree of freedom A depends on the global degree of freedom B.

An earlier version of ALAMODE did not carefully examine the terms in a system to ensure that the global matrix storage was optimal. Instead ALAMODE assumed the worst case for connectivity on an element, and used a dense matrix for the system-level element tangent matrix. This is reasonable for many of the reduced equation models, such as the Fermi and fully-coupled formulations which have element-level flux couplings that depend on all fields involved in the system. A dense element tangent matrix is a poor assumption for fully-kinetic models, which involve a larger number of fields and are dominately reactively coupled. For example, the five species model for phosphorus (Section 6.2) has the following non-zero structure for its element

tangent matrix ([**D**] refers to a dense block and [**d**] refers to a diagonal block).

$$\begin{array}{rccccc} & V & I & PV & PI & P \\ V: & [\mathbf{D}] & [\mathbf{d}] & [\mathbf{d}] & [\mathbf{0}] & [\mathbf{d}] \\ I: & [\mathbf{d}] & [\mathbf{D}] & [\mathbf{0}] & [\mathbf{d}] & [\mathbf{d}] \\ PV: & [\mathbf{d}] & [\mathbf{0}] & [\mathbf{D}] & [\mathbf{0}] & [\mathbf{d}] \\ PI: & [\mathbf{0}] & [\mathbf{d}] & [\mathbf{0}] & [\mathbf{D}] & [\mathbf{d}] \\ P: & [\mathbf{d}] & [\mathbf{d}] & [\mathbf{d}] & [\mathbf{d}] & [\mathbf{d}] \end{array} \tag{5.8}$$

The savings in linear solution computation cost when the equation coupling is optimally handled compared to the case where element tangents are assumed to be dense matrices can be estimated by the global sparse storage required:

$$S \approx \bar{c} n_{nd} n_{nz}^{e} \tag{5.9}$$

where $\bar{c}$ is a proportionality constant, n_{nd} is the number of nodes in a mesh, and n_{nz}^{e} is the number of non-zeros in the element tangent matrix. The optimal storage cost is compared to the costs for dense element tangents for the five species model for phosphorus in the following table for linear simplex element in 1D (2-node "lines"), 2D (3-node triangles), and 3D (4-node tetrahedra).

Dimensionality	Optimal n_{nz}^{e}	Dense n_{nz}^{e}	Cost
1D	48	100	48%
2D	84	225	37%
3D	128	400	32%

5.3 Efficiency comparisons

This section provides comparisons of the efficiency of a dial-an-operator based simulator both from the perspective of the impact of the performance optimizations discussed in this chapter and from the perspective of how well an optimized dial-an-operator simulator performs compared to traditional simulators.

To quantitatively evaluate the impact of the these performance optimizations, two

versions of ALAMODE were profiled using the Unix `gprof` [21] call graph execution profiler. The first version (E) of ALAMODE uses only element-by-element evaluation, implements demand evaluation for expressions without attempting to remove duplicated expressions, and assumes a dense system-level element tangent matrix. Expression evaluation uses the quadrature scheme within a single element to provide the blocks of values to evaluate. The second version (N) uses nodal evaluation where possible, the geometric diffusion discretization, removes duplicated expressions, and optimally determines the structure of the global tangent matrix. Expression evaluation operates on reasonably large blocks of data to ensure that any traversal-related interpretation overhead is fully amortized. The results of this profiling for two different models are discussed in detail in the next two sections.

5.3.1 Profiling a reduced model

The reduced model used for this set of profiling data is a quasi-neutral Fermi model involving two species, boron and arsenic. The model is taken from [29], and involves the following transport equations:

$$\frac{\partial C_{\mathrm{As}}}{\partial t} = \nabla\cdot\left(D_{\mathrm{As}}\nabla C_{\mathrm{As}} + \frac{qD_{\mathrm{As}}C_{\mathrm{As}}}{kT}\nabla\psi\right) \tag{5.10}$$

$$\frac{\partial C_{\mathrm{B}}}{\partial t} = \nabla\cdot\left(D_{\mathrm{B}}\nabla C_{\mathrm{B}} - \frac{qD_{\mathrm{B}}C_{\mathrm{B}}}{kT}\nabla\psi\right) \tag{5.11}$$

where the potential is estimated using a quasi-neutral approximation,

$$\psi = \frac{kT}{q}\,\mathrm{arcsinh}\left(\frac{C_{\mathrm{As}} - C_{\mathrm{B}}}{2n_i}\right). \tag{5.12}$$

The diffusivities are also dependent on the net doping,

$$D_{\mathrm{As}} = D^0_{\mathrm{As}} + D^-_{\mathrm{As}}\left(\frac{n}{n_i}\right) \tag{5.13}$$

$$D_{\mathrm{B}} = D^0_{\mathrm{B}} + D^+_{\mathrm{B}}\left(\frac{n}{n_i}\right)^{-1} \tag{5.14}$$

Task	Time (sec)			Percentage
Linear solution	6.58			5.23%
Time-stepping/Nonlinear	2.07			1.65%
Tangent/Residual	107.99			85.91%
Elementwise	107.99			
Scatter		4.09		3.25%
Tangent		81.36		64.73%
Flux eval.			53.31	42.41%
Flux disc.			28.05	22.32%
Residual		20.05		15.95%
Flux eval.			15.31	12.18%
Flux disc.			4.74	3.77%
Overhead		2.49		1.98%
Nodal	N/A			
Total	125.70			

Table 5.1: Execution time breakdown for simulating a two impurity Fermi model with ALAMODE version E.

where

$$\frac{n}{n_i} = \frac{C_{\mathrm{As}} - C_{\mathrm{B}}}{2n_i} + \sqrt{\left(\frac{C_{\mathrm{As}} - C_{\mathrm{B}}}{2n_i} + 1\right)}. \tag{5.15}$$

The simulations were performed on a 804 point triangulated mesh, leading to a linear system consisting of 1608 equations. The simulation was run for 30 minutes of processing at a temperature was 950^oC. The simulation required 19 time steps and 103 linear solves.

Tables 5.1 and 5.2 summarize the execution profile for each version of ALAMODE.

In these profiling results, the linear solver only includes the time spent preconditioning and iterating to a solution, not times associated with local-to-global scatter operations or other global matrix operations. Time-stepping and nonlinear includes evaluating and assembling a mass matrix, gathering and scattering degrees of freedom between solution vectors and the mesh representation, and operations on the global matrix such as zeroing out before a global assembly and adding quantities to the diagonal to add the temporal discretization the spatially discretized tangent matrix.

Task	Time (sec)			Percentage
Linear solution	6.59			26.14%
Time-stepping/Nonlinear	2.14			8.49%
Tangent/Residual	16.48			65.37%
Elementwise	14.55			57.72%
Scatter		4.58		18.17%
Tangent		6.53		25.90%
Flux eval.			N/A	
Flux disc.			6.53	
Residual		2.11		8.37%
Flux eval.			N/A	
Flux disc.			2.11	
Overhead		1.06		4.20%
Nodal	1.93			7.66%
Scatter		0.00		0.00%
Func. eval.		0.36		1.43%
Field gather		0.92		3.65%
Overhead		0.65		2.58%
Total	25.21			

Table 5.2: Execution time breakdown for simulating a two impurity Fermi model with ALAMODE version N.

Evaluation of the global tangent matrix and residual vector are broken down into elementwise and nodal contributions.

Elementwise contributions are further broken down into the time to scatter element tangent matrices and residual vectors into the global counterparts, the time required to evaluate the element tangent matrices and element residuals, and the time spent in overhead associated with element-by-element evaluation. The overhead includes zeroing out the element tangent and residual, management of the subtangent, and traversal of the elements.

Nodal contributions are further broken down into the time to scatter any system-level diagonal couplings into the global tangent matrix, the time to evaluate all of the expression trees, the time to gather nodal field data from the mesh, and overhead associated with nodal evaluation. This overhead includes the weighting of vector quantities of tangent and residual quantities which were evaluated for use in reactive terms and the management of the nodal evaluation cache.

For this model, the time spent in each of the linear solution, time-stepping, and nonlinear portions of the simulation are almost the same in either version of ALAMODE. This is expected because the system-level element tangent matrix is a dense matrix. Hence, the global tangent would have the same structure in both versions of ALAMODE, and neither the time to alter the matrix for time-stepping nor the time to solve the linear system would be expected to vary. The time to scatter element tangent matrices is also strongly dependent on the number of nonzeros, and the profile data indicates that those times are also roughly the same.

The most significant improvement encountered for this model is in the evaluation of the expressions used by flux terms. Although all of the terms are still evaluated elementwise, version N uses the geometric diffusion discretization and accesses the cached nodal values instead of evaluating expression trees under each term. The time for evaluating expression trees drops from $53.31 + 15.31 = 68.26$ sec. in version E to 1.93 sec. in version N, an improvement factor of 27.

The second most significant improvement is in the evaluation of the flux discretization, itself. In version E, the flux discretization is computed using the consistent finite element weak form, which requires parent-to-spatial mapping of gradients and

a spatial dot product at each quadrature point within an element. In version N, the flux discretization is linearized at the geometric level, and reuses the geometric element flux matrix. The cost of evaluating the spatial discretization drops from $28.05 + 4.74 = 29.79$ sec. to $6.53 + 2.11 = 8.64$ sec.

Although nodal evaluation is listed for version N, the evaluation only computes quantities needed for flux terms. The are no reactive terms in this system. Consequently, there are no matrix scatter operations needed for this system. The amount of time spent in the function evaluation, which is what one would expect to be the computationally intensive part of the nodal evaluation, is actually a very small portion, only 0.36 sec., of the overall execution time. This shows the effectiveness of duplicated expression removal and block evaluation.

There is still some room for further improvements. The most significant speedup would be to change element-by-element tangent and residual routines to compute their residual and all tangent contributions with respect to dependent fields simultaneously. The result would be a reduction in elementwise overhead, and this would eliminate any duplicated evaluation between computation of residual and tangent contributions. Based on these profiling results, this could produce another 35% improvement if all of the flux tangent discretization is considered to be duplicated evaluation. A second source of inefficiency is that the mesh representation uses lists instead of arrays. This may add to execution times with penalties for non-locality of memory accesses, particularly in scatter and gather operations involving degrees of freedom. It is not estimated that this penalty is large, but it may contribute up to 10 percentage points to the total execution time. Even with these remaining inefficiencies, overall simulation time was still improved by almost a factor of five for this model with nodal function evaluation and a geometric spatial discretization.

5.3.2 Profiling a kinetic model

The kinetic model used for this set of profiling data is the five species kinetic phosphorus model presented in Section 6.2. The simulations were performed on a 638 point triangulated mesh, leading to a linear system consisting of 3083 equations. The

Task	Time (sec)			Percentage
Linear solution	324.01			27.47%
Time-stepping/Nonlinear	46.89			3.98%
Tangent/Residual	808.69			68.56%
Elementwise	808.69			
Scatter		237.32		20.12%
Tangent		400.93		33.99%
Reactive eval.			217.48	18.44%
Reactive disc.			120.64	10.23%
Flux eval.			21.75	1.84%
Flux disc.			41.06	3.48%
Residual		110.53		9.37%
Reactive eval.			43.76	3.71%
Reactive disc.			28.20	2.39%
Flux eval.			21.88	1.85%
Flux disc.			16.69	1.49%
Overhead		59.91		5.08%
Nodal	N/A			N/A
Total	1179.59			

Table 5.3: Execution time breakdown for simulating a five species kinetic phosphorus model with ALAMODE version E.

simulation was run for 10 minutes of processing at a temperature of 900^oC. The simulation required 180 time steps and 988 linear solves.

Tables 5.3 and 5.4 summarize the execution profile for each version of ALAMODE. The times for these profiled results are broken down in the same categories as the previous section.

For this model, the cost of the linear solution for version E is 75% greater than that of version N. This difference is due primarily to the optimal sparsity of the global tangent matrix, which reduces the number of structural entries in the matrix by a factor of two. The actual ratio of run times is sublinear (less than a factor of two) because leaving in the structural zeros provides a better ILU preconditioner by effectively providing "fill-in" locations. The better preconditioner reduces the average number of iterations needed for each iterative linear solution. Version E averages 2.7 iterations for each linear solution, and verion N averages 5.3 iterations for each linear

Task	Time (sec)			Percentage
Linear solution	185.39			52.80%
Time-stepping/Nonlinear	28.97			8.25%
Tangent/Residual	136.75			38.95%
Elementwise	101.76			28.98%
Scatter		44.15		12.57%
Tangent		11.78		3.36%
Flux eval.			N/A	
Flux disc.			11.78	
Residual		10.85		3.09%
Flux eval.			N/A	
Flux disc.			10.85	
Overhead		34.98		9.96%
Nodal	34.99			9.97%
Scatter		10.85		3.09%
Func. eval.		1.74		0.50%
Field gather		11.74		3.34%
Overhead		10.66		3.04%
Total	351.11			

Table 5.4: Execution time breakdown for simulating a five species kinetic phosphorus model with ALAMODE version N.

solution. Since both the preconditioning and per-iteration costs are greater with the version E's suboptimal sparse storage, the overall linear solution cost is still greater.

The time spent under the time-stepping and nonlinear portions has also larger for version N because the larger number of structural zeros increases the cost to zero out the global matrix and to add in the diagonal components from the time-stepping discretization. The size of the global matrix also increases the cost of both the elementwise scatter, 237.32 sec. vs. 44.15 sec., and the overhead of the elementwise evaluation, 59.91 sec. vs. 34.99 sec. in version E.

As was the case with the previous set of profiling results, the most significant improvement in total time encountered for this model is in the evaluation of expressions. In this model, however, there are both reactive terms and flux terms. If these are lumped together, the cost of evaluating expression drops from 299.68 sec. in version E to under 24.14 sec$\dot{}$(total nodal time minus nodal scatter), an improvement of more than a factor of twelve.

Evaluation of the discretization also shows significant improvements. Reactive terms were eliminated from element-by-element evaluation. In doing so, the contribution of reactive discretization was reduced from 148.84 sec. to under 21.51 sec. (nodal scatter plus nodal overhead). The cost of the flux discretization also improved by about a factor of three, about the same as the improvement in the reduced model.

Again, there is still room for improvement in the element-by-element flux discretization and in the degree of freedom gather and scatter. Even with these remaining inefficiencies, overall execution time was improved by a factor of 3.3. Further optimizations may only be able to improve this another 10-20%.

5.3.3 Comparison with other simulators

While these results show that most of the inefficiencies that can be attributed to a dial-an-operator implementation have been eliminated with the techniques discussed in this chapter, they do not provide a comparison with how well a dial-an-operator simulator performs against traditional "hand-coded" simulators. Although it would be best to compare the efficiency on a variety of models, existing traditional simulators

typically only implement one or two different formulations. Implementation of new handcoded models would be too time consuming to justify solely for the purpose of performance benchmarking. Only widely implemented diffusion formulations, Fermi and fully-coupled, were considered for comparison.

To ensure that the comparison is as fair as possible, it would be nice to have a guarantee that each simulator actually implements a correctly linearized discretization of the equations which it claims to implement. With the dial-an-operator approach, the correctness of the linearization and discretization for the system of equations can be implied from the correctness of each individual library component. Because each library component is relatively simple, each component is easily verified.

Unfortunately, correctness is virtually impossible to guarantee for the hand-coded simulators. One known flaw is that SUPREM's implementation of the fully-coupled model does not compute a consistent tangent by differentiating all of the quantities associated with all of terms [22]. While this may have initially started out as a model-specific optimization, the developer ignored what were believed to be insignificant contributions to the tangent matrix based on a specific range of input data, the inconsistent tangent later led to convergence problems when the range of input data changed. Consequently, only performance based on the Fermi model will be compared, as the simpler Fermi model has been better validated, at least in SUPREM.

For this performance benchmark, the Fermi model was simulated for two impurities, boron and arsenic, on identical 2D structures consisting of approximately 1000 nodes (2000 discretized equations) and 1600 triangles. Identical model parameters and error control parameters were specified, where possible. Simulation run times and other performance-related metrics are shown in Table 5.5.

The primary objective of this comparison is to determine how well an optimized dial-an-operator implementation of discretization performs when compared against traditional hand coded implementations, and several performance metrics that could be used as the basis for this comparison. Overall run time is attractive because this is the performance metric that most end users of a simulator care about, but this is more strongly influenced by the measures of error that are used to detect nonlinear convergence and to control the time-step than by the efficiency of the equation

	Stanford SUPREM (v9305)	ALAMODE	FLOOPS ('96 release)
run time (sec)	78.8	31.3	74.8
# time steps	27	15	14
# nonlinear steps	356	125	168
average nonlinear/time step	13.2	8.33	12.0
evaluation and linear solve (sec)	0.22	0.25	0.45
% time in linear solve	44.5	28.1	64.3
discretized eval. and scatter (sec) per assembly/linear solve	0.12	0.18	0.16

Table 5.5: Run-time performance of ALAMODE, SUPREM, and FLOOPS on a FERMI diffusion model.

discretization.

The average number of nonlinear iterations needed to solve each time step has almost a factor of 2 difference between the best and worst simulators. This large difference is due, at least in part, to the inability to specify the nonlinear convergence parameters in SUPREM and FLOOPS. In both of these simulators, the values for absolute and relative convergence criteria in both the residual norm and the update norm are hardcoded, and were not changes for these simulations. Also, the nonlinear convergence behavior of FLOOPS indicates a possible error in the linearization of the Fermi model. Convergence is usually detected using the update norm, and the final residual norm is either larger than or not significantly reduced from the initial residual norm for a large percentage of nonlinear solves. Even with this questionable convergence behavior, the simulation results from FLOOPS show the same evolution of dopant profiles as ALAMODE and SUPREM to within reasonable error bounds.

The number of time steps also varies significantly among the simulators. The time-stepping behavior of the three simulators is shown in Figure 5.7. All three use the Milne's device to estimate temporal error in the current time-step and use that error estimate to select a reasonable interval for the next time-step. Both ALAMODE and FLOOPS limit increases in stepsize (Equation 3.63), but SUPREM places no bounds on how much the stepsize increases. Although one would expect that this lack of damping would allow SUPREM to complete the temporal integration with a

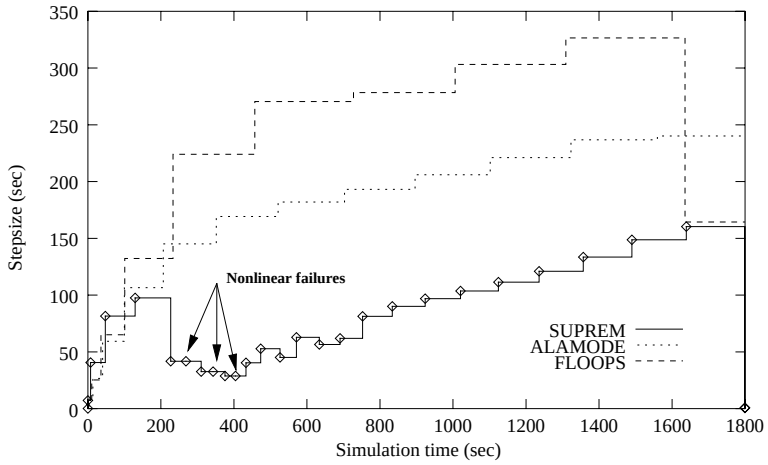

Figure 5.7: Time-stepping behavior of ALAMODE, FLOOPS, and SUPREM on the Fermi model.

smaller number of time-steps, SUPREM actually requires almost twice as many time-steps as ALAMODE or FLOOPS. The exact cause of this anamoly is not known, but SUPREM experiences several failures in nonlinear solution of a trapezoidal rule sub-step, which would normally indicate that overshoot due to too large of a time-step has driven concentrations negative. Overshoot seems unlikely because both ALAMODE and FLOOPS successfully integrate the same interval with larger time-steps without encountering negative concentrations. SUPREM's time-steps are also not monotonically increasing as they are in both ALAMODE and FLOOPS. Regardless of what the underlying reasons are for the differing behavior, those differences should be eliminated from the metric used to compare discretization efficiency.

Instead of overall run time, a better metric would be the time to evaluate and solve a discretized and linearized system. Unfortunately, this metric is also a little misleading in that it is influenced by the performance of the linear solver, and each of these simulators use different packages for linear solution. ALAMODE and SUPREM use different linear solution packages, and SUPREM's solver is likely to be less expensive than ALAMODE's for the well-conditioned matrices produced by the discretized Fermi model [30]. Although both ALAMODE and FLOOPS use an incomplete lower/upper (ILU) factorization for preconditioning and a stabilized biconjugate

gradient (BCGS) iteration, ALAMODE uses a canned package (PETSc-SLES [2]), and FLOOPS contains a custom hand-coded implementation. To accurately gauge only the discretization efficiency, the cost of linear solution should be eliminated as a factor in the comparison. With the differences in the cost of linear solution removed, the dial-an-operator implementation of discretization in ALAMODE approaches that of a hand-coded implementation. In the worst case, ALAMODE's evaluation of the discretization was only 50% slower than a hand-coded simulator. If ALAMODE were modified to compute the tangent matrix simultaneously with the residual, this gap would be significantly reduced.

Further decomposition of the run time to isolate only the evaluation of the spatial discretization is not possible because SUPREM and FLOOPS are not structured in a way that permits `gprof` to measure the CPU times attributed to local-to-global matrix scatter, time-stepping adjustments to the global tangent matrix and residual, or convergence tests for the nonlinear solution.

5.4 Conclusions

This chapter discussed implementation issues for *dial-an-operator* evaluation of the discretization of user-specified PDEs. The implementation of the discretized terms in some ways resembles traditional codes in that the "element" routines consume mesh elements and produce residual vectors and tangent matrices, but a key extension is the ability to plug arbitrary expressions into these routines. The expressions are represented in a tree and provide both the values needed to compute the residual and the values and derivatives needed to compute an analytic tangent.

Implementation issues related to efficient evaluation of these expressions were also presented. For efficient evaluation on modern pipelined computers, each node should evaluate itself on a block of data, both to minimize interpretation overhead and to increase the amount of instruction-level parallism available to a compiler. Duplicated evaluation can be eliminated by recognizing that nodes are shared among elements and by detecting common subexpressions in the expression trees used by a system of equations. Run-time profiling shows that, with these efficiency enhancements, the

cost of evaluating the expressions and their derivatives is small compared to the cost of a flux discretization or the cost linear solution. A comparison with hand-coded implementations shows that near hand-coded efficiency can be obtained with *dial-an-operator* implementation.

Chapter 6

Model Examples

TCL [41] was chosen as the language for both representation of the model and control of the simulation. The TCL-based model representation mirrors the information model discussed in Chapter 2 and provides commands to build models out of the object-oriented TCL commands: `model`, `systemInRegion`, `equation`, `operator`, and `function`. Global TCL variables are used as parameters, and the `model` command provides for specification of fixed (Dirichlet) boundary conditions and field continuity constraints across material interfaces.

It is best to build a model from the bottom up. TCL variables that will be used as field name aliases and model parameters are defined first. Then ALAMODE functions, operators, equations, and systems are defined, and the model is specified. After the model is specified, it can be simulated on a device structure by calling the appropriate solve option from the model's instance.

Although the examples presented here show a static TCL representation of a complete diffusion model, it is also possible to dynamically build the equations, either based on user-supplied input or based on the available fields and regions within a specific structure. Field and region query commands are available to provide access to the latter.

Two representative diffusion models are presented below. The first model contains equations that describe diffusion of a single dopant in multiple materials with a Fermi-level diffusivity in the semiconductor and constant diffusivity in the insulator.

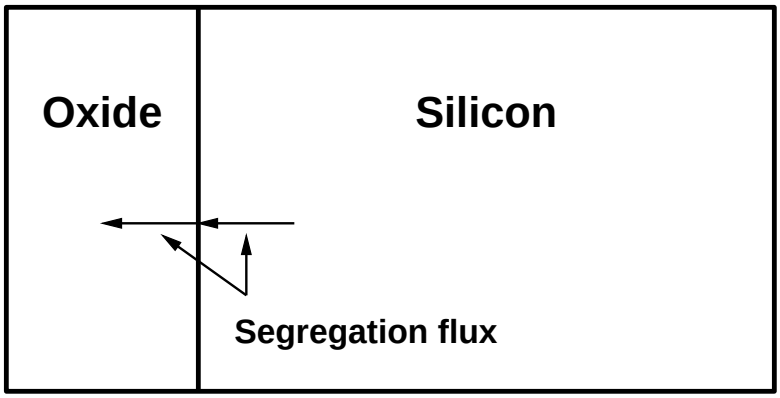

Figure 6.1: Materials and segregation fluxes

Transport of the dopant across the interface is described with a segregation flux term. This model demonstrates the specification and use of functions and how systems of equations are coupling among regions. The second model is a fully-kinetic formulation which considers pairing reactions as well a diffusive transport of native defects and dopants paired with native defects. This model demonstrates specification and solution of multiple equations in a single region and specification of external boundary conditions.

6.1 Multi-region diffusion with segregation

This example demonstrates a model for diffusion in two regions (silicon and silicon dioxide) with a segregation interface flux between the regions (Figure 6.1). In the silicon region, there is a single dopant diffusion equation for boron (B):

$$\frac{\partial C_{\mathrm{B}}}{\partial t} = \nabla \cdot D_{\mathrm{B}} \nabla C_{\mathrm{B}} + \nabla \cdot D_{\mathrm{B}} C_{\mathrm{B}} \nabla \ln \left(\frac{p}{p_i} \right) \tag{6.1}$$

If there is only a single dopant present and if complete activation is assumed, the

diffusivity is given by

$$D_{\mathrm{B}} = D^0 + D^+ \left(\frac{p}{p_i} \right) \tag{6.2}$$

where

$$\frac{p}{p_i} = \frac{C_{\mathrm{B}}}{2n_i} + \sqrt{\left(\frac{C_{\mathrm{B}}}{2n_i} \right)^2 + 1} \tag{6.3}$$

The diffusion equation in the silicon dioxide region is given by a simplified form:

$$\frac{\partial C_{\mathrm{B}}}{\partial t} = \nabla \cdot D_{\mathrm{B}} \nabla C_{\mathrm{B}} \tag{6.4}$$

where D_{B} is a constant because the species does not become electrically active in the amorphous, insulating material.

There is also a transport flux across the silicon/oxide interface which is approximated by

$$\text{flux} = Tr \left(C_1 - C_2/m \right) \tag{6.5}$$

where C_1 and C_2 are the concentrations of C_{B} on either side of the interface.

Evaporation of the dopant at the exposed silicon dioxide surface is assumed to be negligible, as is any flux transport at the back extent of the structure.

6.1.1 The model script

The first few lines provide aliases for field names. Although aliasing field names is not necessary, it does create more portable and better documented model descriptions. Because some structure file formats (e.g. SUPREM) can only represent a fixed set of fields, it is often necessary to use an unused impurity (e.g. gold) to represent another quantity of interest (e.g. {311} clusters).

```
set BinSi BoronInSilicon
set BinOx BoronInOxide
```

The next group of lines define the parameters (global TCL variables) that will be used in the model. A TCL `proc` is used to evaluate the Arrhenius temperature dependence.

```
# useful constants
set zero 0.0
set one 1.0

set tempC 1000

set tempK [expr $tempC + 273]
set kT    [expr 0.0259 * $tempK / 300]

proc Arrh {pre Ea} {
    global tempK
    global kT
    return [expr $pre * exp(-$Ea / $kT)]
}

#   Diffusivities are in cm^2/s
#   Activation energies are in eV
#   Transport coefficients are in cm/s

set dbsi0 [Arrh 0.037 3.46]
set dbsip [Arrh 0.76 3.46]
set dbox [Arrh 3.15e-4 3.53]
set Trn -1.66e-7
```

```
set Seg 2

# Intrinsic carrier concentration
set ni0 3.9e16
set niE 0.605
set niP 1.5
set ni [expr [Arrh $ni0 $niE] * pow($tempK,$niP)]
```

The functions that will be used in the model evaluation are defined next. ALAMODE's high-level template functions, `carrierConc` and `fermi`, are used for the expressions in (6.3) and (6.2).

```
# if the dopant is not completely active, change the following line
set netActive $BinSi
set carrierConc [function carrierConc \
                    -parameters [list $netActive ni]]

# Use the Fermi diffusivity function
# for boron diffusivity in the silicon.
set DBSi [function fermi -parameters \
            [list $carrierConc dbsi0 dbsip zero]]
```

Next, the operators, equations, and systems are created for each region. In the silicon region, the single equation in the system of equations is composed of two terms, one for diffusive effects and one for electric field effects.

```
set BSiDiffOp [operator diffusion -sign 1 \
                  -parameters [list $DBSi $BinSi] ]

set BSiEFOp [operator quasiFermiEField -sign 1 \
                -parameters [list $BinSi $DBSi $carrierConc] ]
```

```
set BSiEq [equation −parabolic −solvesFor $BinSi \
              −operators [list  $BSiDiffOp $BSiEFOp ] ]

set siliconSystem  [systemInRegion −region Silicon \
                       −equations [list  $BSiEq] ]
```

In the oxide region, the single equation in the system of equations contains only a diffusive term.

```
set BOxDiffOp [operator diffusion −sign 1 \
                  −parameters [list dbox $BinOx] ]

set BOxEq [equation −parabolic −solvesFor $BinOx \
              −operators [list  $BOxDiffOp] ]

set oxideSystem [systemInRegion −region Oxide \
                    −equations [list  $BOxEq] ]
```

The interface flux constraints are written as an inflow and outflow equation for the two materials in the interface region. With the segregation flux, the dopant is transported across the interface without trapping of dopant at the interface. More advanced interface flux transport, including the trapping and release of dopants in the interface, can easily be implemented by using transient equations in interface regions.

```
set BSegOutOp [operator segregation −sign 1 \
                  −parameters [list $BinSi $BinOx Trn Seg] ]
set BSegInOp [operator segregation −sign −1 \
                  −parameters [list $BinSi $BinOx Trn Seg] ]

set BSegOutEq [equation −solvesFor $BinSi \
                   −operators [list  $BSegOutOp] ]
set BSegInEq [equation −solvesFor $BinOx \
```

```
                    -operators [list $BSegInOp] ]

set interfaceSystem [systemInRegion -region Oxide/Silicon \
    -equations [list $BSegOutEq $BSegInEq ]]
```

Finally, the three systems are combined into a single model.

```
set modelDiff [model -systems [list \
  $siliconSystem $oxideSystem $interfaceSystem ] ]
```

At this point, there is a complete script representation of model. This model can then be applied to a device by loading a structure and using the solve command to run the model as shown below.

```
sstructure -infile implant.str

# Stop the simulation after 3600 seconds
$modelDiff -stopTime 3600

$modelDiff -transientSolve
sstructure -outfile final.str
```

6.1.2 Simulation results and discussion

The initial and diffused profiles for a 1 hour 1000^oC anneal are shown in Figure 6.2. The parameter values at this temperature are:

$$\begin{aligned} \text{Si: } D_{B0} &= 7.86 \times 10^{-16} cm^2/s \\ \text{Si: } D_{B+} &= 1.61 \times 10^{-14} cm^2/s \\ \text{Si: } p_i &= 7.2 \times 10^{18} cm^{-3} \\ \text{Ox: } D_B &= 3.54 \times 10^{-18} cm^2/s \\ \text{Si/Ox: } h &= 1.67 \times 10^{-7} cm/s \\ \text{Si/Ox: } m &= 2 \end{aligned}$$

The diffused profile is identical to that predicted by the Fermi model in SUPREM, and shows the essential characteristics of this model:

1. the development of the concentration discontinuity at the material interface due to the segregation fluxes,

2. a flattened peak concentration caused by a higher diffusivity in the extrinsic device area (where $p > p_i$), and

3. almost no impurity movement in the oxide region, where the diffusivity is small.

This simulation was performed on a 312 point SUPREM-IV mesh structure (314 degrees of freedom). With equivalent error control parameters ALAMODE took 44 times steps to integrate the equation and completed in 6.3 seconds. Stanford SUPREM-IV (version 9130) took 49 time steps and completed in 11.4 seconds. The main reasons for ALAMODE's better performance on this example are:

- SUPREM computes additional quantities that are used by other models, which are not necessary for the Fermi model, while ALAMODE only computes the quantities that are used for the discretized equations in the given model. The discretized models in SUPREM reuse code as much as possible between the different diffusion models, and the code design chose to accept an efficiency

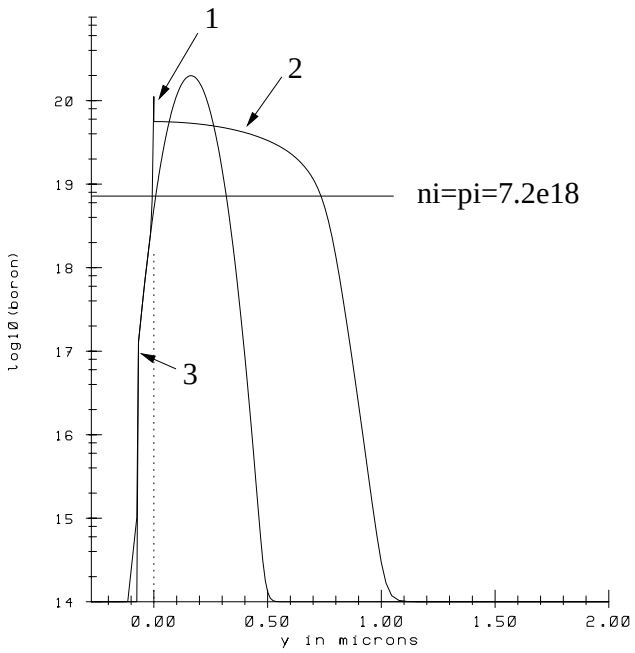

Figure 6.2: Initial and final boron profiles in the silicon and silicon dioxide regions

penalty when evaluating simpler models in order to maximize code reuse for similar formulations.

- SUPREM exhibits slower nonlinear convergence during each time step, hence SUPREM needs a much larger number of matrix assembly and linear solves. The slower convergence is likely caused by a decoupled nonlinear solution in SUPREM. The self-consistent nonlinear system is split into two updates, one that updates potential in terms of impurity concentrations and another that updates the impurity concentrations based on the time step and potential. The two updates are alternated until the convergence criteria are satisfied.

6.2 Multi-species reactive-diffusive model

This example demonstrates a strongly-coupled, highly-reactive model for phosphorus diffusion [49]. The equations in the silicon region model the following physical effects:

- creation and annihilation of isolated silicon vacancies and interstitials ($I+V \leftrightarrow$)
- pairing and dissociation reactions involving an isolated phosphorus atom and either a silicon vacancy or silicon interstitial ($P+V \leftrightarrow E$ and $P+I \leftrightarrow F$)
- migration of silicon interstitials, vacancies, and paired species via a diffusive mechanism

The equations are:

$$\begin{aligned}
\frac{\partial V}{\partial t} &= \nabla \cdot D_V \nabla V - k^E_{for} PV + k^E_{rev} E - k_{bi}(VI - V^{eq} I^{eq}) \\
\frac{\partial I}{\partial t} &= \nabla \cdot D_I \nabla I - k^F_{for} PI + k^F_{rev} F - k_{bi}(VI - V^{eq} I^{eq}) \\
\frac{\partial E}{\partial t} &= \nabla \cdot D_E \nabla E + k^E_{for} PV - k^E_{rev} E \\
\frac{\partial F}{\partial t} &= \nabla \cdot D_F \nabla F + k^F_{for} PI - k^F_{rev} F \\
\frac{\partial P}{\partial t} &= -k^E_{for} PV + k^E_{rev} E - k^F_{for} PV + k^F_{rev} F
\end{aligned}$$

The parameter values are also taken from [49] for anneals performed at 900^oC.

	C_{eq}	D_{def}	D_{pair}	k^{pair}_{for}	k^{pair}_{rev}	k_{bi}
	(cm^{-3})	(cm^2/s)	(cm^2/s)	(cm^3/s)	(s^{-1})	(cm/s)
V	10^{14}	10^{-10}	10^{-13}	10^{-14}	10	10^{-10}
I	10^{14}	10^{-9}	2×10^{-13}	10^{-14}	12	10^{-10}

6.2.1 The model script

Again, the script begins with aliasing of the field names.

```
set V VacancyInSilicon
set I  InterstitialInSilicon
set P PhosphorusInSilicon
set E PVpairInSilicon
set F PIpairInSilicon
```

Specification of the parameters as global TCL variables is very similar to the previous example

```
set Dv 1.0e−10
set Di 1.0e−9
set De 1.0e−13
set Df 2.0e−13

set Kfe 1.0e−14
set Kre 10.0
set Kbi 1.0e−10
set Kff 1.0e−14
set Krf 12.0

set Vstar 1e14
set Istar 1e14
```

Each of equations in this system are very similar to each other, hence only the ALAMODE script fragment for the equation for vacancies is shown below. This equation is built from three terms, a `diffusion` term, a `k1fg_k2h` term which models the three species reaction, and a `bimole` term for the bimolecular recombination/generation.

```
set vacancyEquation \
  [equation −parabolic \
   −solvesFor $V \
   −operators [list \
       [operator diffusion −sign 1 \
         −parameters [list Dv $V] ] \
       [operator k1fg_k2h −sign −1 \
         −parameters [list $P $V $E Kfe Kre] ] \
```

```
    [operator bimole -sign -1 -parameters \
      [ list  $I  $V Kbi Istar Vstar ] ] ] ]
```

The five equations are combined into a single system in the silicon region, and this system is the only system in the model.

```
set modelDiff [model \
  -systems [list \
    [systemInRegion -region Silicon \
      -equations [list \
        $vacancyEquation \
        $interstitialEquation  \
        $pvEquation \
        $piEquation \
        $pEquation  ] ] ] ]
```

The model is used to simulate a gas phase predeposition step where the surface concentration of phosphorus is specified. The boundary conditions needed are Dirichlet, which fix the values of P, I, and V at the exposed surface and are specified at the model level as a list of field/region pairs. Adjustments to the boundary conditions can be added to a model at any time prior to the simulation.

```
$modelDiff -fixedBC [list \
    [ list  $V ExposedSilicon] \
    [ list  $P ExposedSilicon] \
    [ list  $I ExposedSilicon] ]
```

Unlike the previous example, where the initial field values were extracted from the structure file, the field values in the structure file are discarded by overwriting them with flat profiles after the structure is loaded.

```
structure - infile   init . str
```

```
field  $V −setValue $Vstar
field  $I −setValue $Istar
field  $E −setValue 1.0
field  $F −setValue 1.0
field  $P −setValue 1.0
```

Before simulating this predeposition step, the surface concentration is initialized to the minimum of the dopant concentration in the ambient and the solid solubility concentration value.

```
set SurfConc 3.2e20

field  $P −inRegion ExposedSilicon −setValue $SurfConc
```

6.2.2 Simulation results and discussion

The time evolution of the profiles for each species is shown in Figure 6.3. These profiles show the ability of the coupled equations to correctly predict the characteristic kink and tail that is observed experimentally for phosphorus.

The same model script was also applied to a predeposition simulation that uses a 2D masked structure. Other investigators ([48] and [42]) applied this model to similar structures and reported high failure rates in the iterative linear solution as the time step increased beyond 0.05 to 1.5 sec. due to an increasingly poor condition number of the tangent matrix. In [48], when the iterative linear solve failed, the time step was reduced, improving the condition of the tangent matrix, and the time interval was retried. In [42], a direct factorization was used when the iterative linear solve failed. Neither of these approaches to overcome numerical difficulties are desirable for a production code that may be required to simulate similar models on large 2D or 3D structures. Reducing, and effectively capping, the time step can lead to prohibitively expensive simulation times if the transient interval for simulation is large. Direct

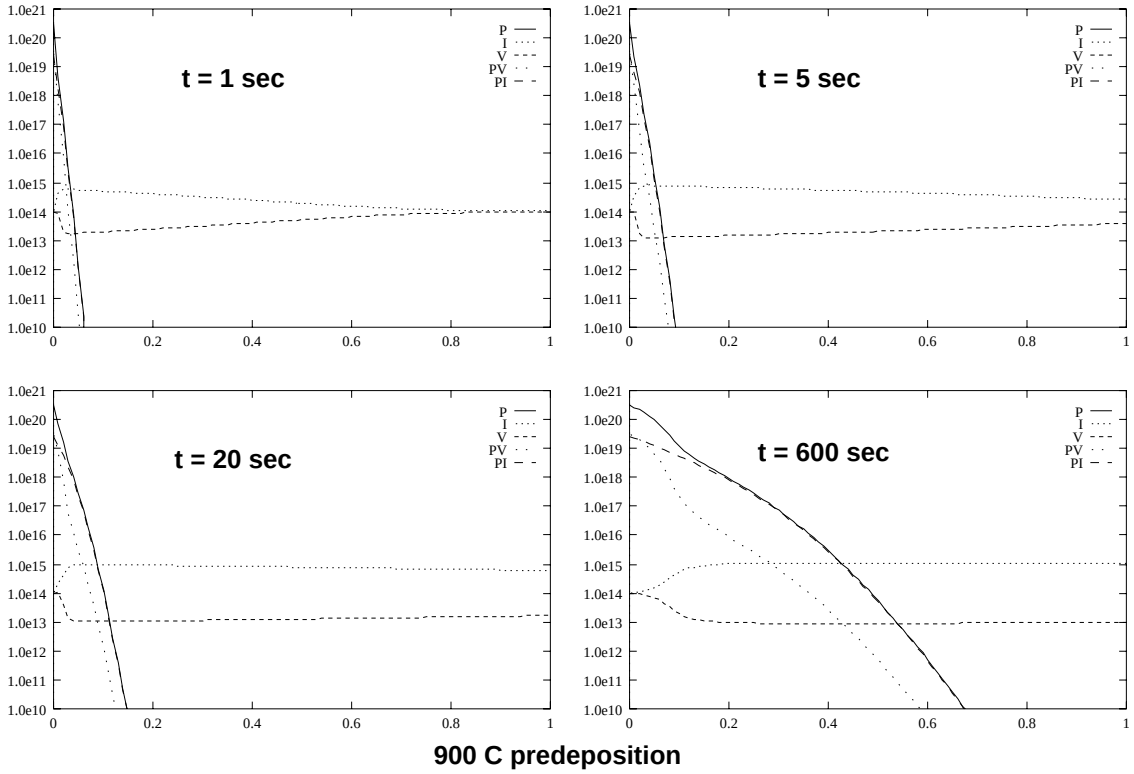

Figure 6.3: Evolution of immobile phosphorus, interstitials, vacancies, phosphorus-interstitial pairs, and phosphorus-vacancy pairs at 900^oC

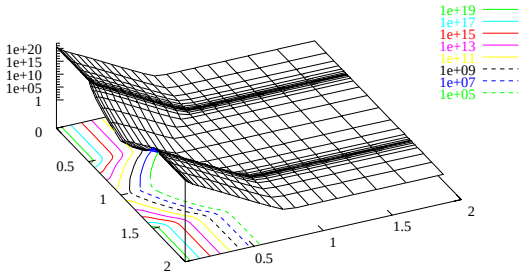

Figure 6.4: Total phosphorus concentration after a 1 minute predeposition at 900^oC on a 25×25 mesh.

factorization may not even be possible for large 2D or 3D structures when simulated on a machine with a limited amount of memory.

ALAMODE's iterative linear solver uses an incomplete LU factorization for preconditioning, and has proven to be robust for this model on 2D structures. Linear solution iteration counts remained below 20, even with time steps of up to 70 seconds. Although supported, no additional fill-in was required, even when using the optimal sparse storage for the global tangent matrix.

Although this model is somewhat dated from the perspective of current research into dominant physical effects, current diffusion modeling is moving more toward formulations that treat species reactions kinetically. This is a stark contrast to past approaches where the model was first simplified by assuming that certain reactions are in local equilibrium, and equations for many species were eliminated using a relationship that provided the concentration of a given species in terms of the local concentrations of other species. From a model implementation perspective, the most significant implication of this change is that these kinetic models no longer fit into a paradigm derived for implementing variations on the fully-coupled formulation, such as the implementation frameworks provided by SUPREM and FLOOPS.

6.3 Growth and evolution of {311} defects

As the temperatures and times for thermal annealing continued to be reduced in order to maintain reasonable junction movement ($\Delta x_j \sim \sqrt{D_{\text{eff}}\, t}$) for shrinking device

dimensions, process engineers began to notice a transient behavior where the effective diffusivity was significantly greater than that predicted by equilibrium models, especially after an implant step. The effective diffusivity could up to several thousand times that predicted by the equilibrium model. Understanding the phenomenon of transient enhanced diffusion and developing models that were predictive over new ranges of annealing conditions quickly became a high priority.

A brief history of the development of models for transient diffusion is covered in [45]. The underlying cause of the transient diffusion is crystal damage due to an implant. Although a large number of both interstitial and vacancy point defects are created when implanted ions knock atoms off their lattice sites, those defects recombine quickly, leaving an excess interstitial defect for each implanted atom. Those excess interstitials quickly form rod-like extended defect structures ({311} defects) in channels in the lattice. The {311} defects dissolve at a slower rate, and the interstitials that they emit enhance the diffusivity of impurities, such as boron, that migrate primarily by paring with an interstitial defect.

The simplest models for trapping and release of interstitials by {311} defects only consider the total number of interstitials trapped in {311} defects [44]:

$$\frac{\partial C_{311}}{\partial t} = 4\pi\alpha a D_I I C_{311} - C_{311}\left(\frac{D_I}{a^2}\right)\exp\left(-\frac{E_b}{kT}\right) \tag{6.6}$$

where α is the capture radius, a is the average inter-atomic spacing, and E_b is a cluster binding energy. The addition of this single equation is sufficient to predict enhancements for situations where the interstitial transient due to {311} dissolution runs to completion at a uniform temperature, but it is not adequate to predict enhancements during anneals involving a temperature ramp or where the {311} defects do not completely dissolve.

To properly model the impact of {311} clusters under these conditions, it is necessary to include the effects of Ostwald ripening [7] and simulate independent equations for each {311} defect size. The model, as proposed by Saito [51], considers the growth

and dissolution of a {311} defect of size n by the addition or release of a single interstitial,

$$C_n + I \underset{k_n^r}{\overset{k_n^f}{\rightleftharpoons}} C_{n+1}, \tag{6.7}$$

where

$$k_n^f = 4\pi a D_I, \quad k_n^r = \frac{D_I}{a^2} \exp\left(\frac{E_{(n)}}{kT}\right). \tag{6.8}$$

The binding energy ($E_{(n)}$) for the evaporation rate, depends on the number of self-interstitials in {311} defects [44]. This model for the evolution of {311} defects is incorporated into the following system of coupled diffusion-reaction equations:

$$\begin{aligned} \frac{\partial I}{\partial t} &= D_I \frac{\partial^2 I}{\partial x^2} + \frac{\partial}{\partial x}\left(\frac{D_B I}{I^*} \cdot \frac{\partial B}{\partial x}\right) + \frac{\partial}{\partial x}\left(\frac{D_B B}{I^*} \cdot \frac{\partial I}{\partial x}\right) \\ &- \sum_n \frac{\partial C_n}{\partial t} - k_r \left(IV - I^* V^*\right), \end{aligned} \tag{6.9}$$

$$\frac{\partial V}{\partial t} = D_V \frac{\partial^2 V}{\partial x^2} - k_r \left(IV - I^* V^*\right), \tag{6.10}$$

$$\frac{\partial B}{\partial t} = \frac{\partial}{\partial x}\left(\frac{D_B I}{I^*} \cdot \frac{\partial B}{\partial x}\right) + \frac{\partial}{\partial x}\left(\frac{D_B B}{I^*} \cdot \frac{\partial I}{\partial x}\right), \tag{6.11}$$

$$\frac{\partial C_n}{\partial t} = k_{n-1}^f I C_{n-1} - k_n^r C_n - \left(k_n^f I C_n - k_{n+1}^r C_{n+1}\right). \tag{6.12}$$

Equation 6.12 is repeated for each different defect size, $n \geq 2$, up to sizes that, because of their small concentration, do not contribute significantly to the total stored interstitial population, typically $30 < n_{\max} < 120$.

Figure 6.5 shows typical simulation results that can be achieved with this model as originally reported in Reference [51]. A superlattice experimental structure is used to detect the transient supersaturation and diffusion of free interstitials. In this simulation, {311} defects of 31 different sizes were included in the model. The final simulation time exceeds the dissolution time for the {311} defects, but [51] shows an excellent match with SIMS profiles taken from the wafers at various anneal times

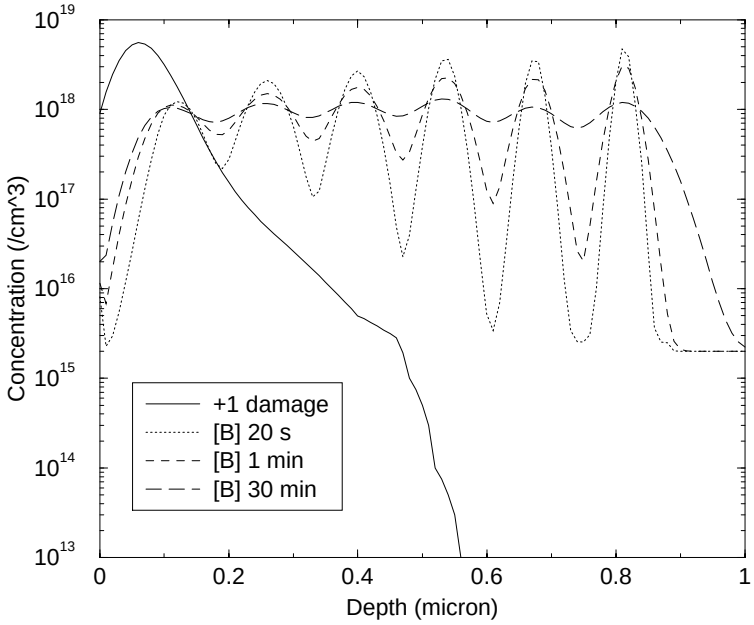

Figure 6.5: Simulation results for a superlattice sample implanted with $5 \times 10^{13}\text{cm}^{-2}$ Si^{+} at 50keV and annealed at $820^{o}C$.

both before and after the defects dissolve and the transient enhancement finishes.

Although the implications of this and similar physical models are exciting because they provide the ability to accurately predict profile evolution under annealing conditions dominated by transient enhancements, one other set of statistics deserves note. The complete description of the 34 model equations consists of fewer than 250 lines of TCL code with another 500 lines of TCL to perform the computation of all of the numerical parameters. Both of these could be significantly reduced by using a table-driven approach to computing parameters and creating equations.

6.4 Summary

This chapter presented several examples of diffusion models that have been implemented and simulated in ALAMODE. The first model specifies a single equation for each impurity and uses diffusivities that depend on the local Fermi level, which in turn depends on the net active carrier concentration. The implementation of this

example demonstrates the several key capabilities of ALAMODE, including: automatic differentiation (of the diffusivities), support for different systems of equations in different regions, and global coupling through an interface segregation flux.

The second model is a fully kinetic description of phosphorus diffusion which considers five species: mobile silicon interstitial and vacancies, mobile pairs of phosphorus with interstitials and vacancies, and unpaired immobile phosphorus. The pairing and recombination reactions are very strong and the diffusivities vary over several orders of magnitude. This results in a stiff system of equations to be integrated. The TR/BDF2 integration technique is able to efficiently integrate this system with time-steps controlled by estimates of the temporal error.

The third model is representative of the complexity of models used in current research into the mechanisms behind transient enhanced diffusion during quick low temperature anneals. This model provides kinetic treatment of the growth and dissolution of individually sized {311} defects. Such treatment is needed to properly model the effects of Ostwald ripening in which the average size of a defect increases and the interstitial release rate decreases with the anneal time.

Although full kinetic treatment it is not always necessary, the trend in diffusion model development is toward more kinetic treatment, whether it is for the development of extended defects or smaller impurity-defect clusters. The equations for these models are significantly more stiff than the previous generations of reduced equation models (Fermi and fully-coupled), and there is a large number of independent variables that must be solved simultaneously. ALAMODE has proven itself as an efficient and robust simulation environment for these advanced systems of equations.

Chapter 7

Conclusions and Recommendations

The continuing reduction in semiconductor device sizes and corresponding reduction in the times and temperatures required for thermal annealing temperatures has produced a need for simulation environments that support very rapid prototyping of models for the migration of impurities during thermal processing. A second simulation need is driven by the fact that device structures are becoming more 3-dimensional in nature; hence, such a rapid prototyping should support simulation of those models on 1D structures, which are primarily used for model development and calibration, on 2D structures, which is still adequate for most "core" device-level electrical simulation, and on 3D structures for extraction of 3D parasitics and true 3D effects such as the reverse short-channel effect.

To address these requirements, a simulation program, ALAMODE, has been developed. ALAMODE has the ability to take a formal description of partial differential equations and to discretize and solve those equations on a meshed device structure of any spatial dimensionality. The description mechanism for the PDEs is based on a dial-an-operator approach and provides a library of discretized "operators" as well as a library of functional relationships that can be used to build arbitrary functional expressions. A well-defined programming interface is also provided as a mechanism to interface with external meshed semiconductor wafer representations in a variety of formats.

7.1 Summary

The implementation of such a rapid prototyping environment requires the development of a consistent and encompassing information model suitable for representation of realistic equation-based diffusion models, robust numerical discretization techniques, and efficient techniques for converting the information model into an executable representation.

The development of an information model for representation of diffusion models was discussed in Chapter 2. The information model considers the relationships among a relatively small set of well-defined objects that structure a diffusion model into a number of coupled systems of equations in regions and an optional set of dirichlet/continuity constraints. Each system of equations is composed of a number of equations, and each equation is used to solve for a specified field in a material. The equations are assembled using a dial-an-operator paradigm as a sum of specific instances of operators or discretized terms that are provided by a library of reusable components. To facilitate reuse of operators, they make use of replaceable scalar quantities, which are either native interpolated field variables or a functional transform of other scalar quantities. The information model also defines how these model description objects relate to an abstract meshed representation of a device structure.

Chapter 3 provides background on the numerical techniques used for discretization. A semi-discrete finite element approach was chosen over the finite volume approach which has been more widely used by the TCAD community. One of the motivations for this choice was an assumed superiority for FEM techniques which are usually backed with a better mathematical foundation than ad hoc discretizations, at least for structural mechanics and fluid dynamics applications. Unfortunately, the promise of superiority did not hold given the criteria of robust solution of discretized reactive-diffusive equations.

To guarantee an attainable discrete solution of these equations, the discretized system of equations must maintain a reactive-definite property. For an arbitrary mesh that is not subject to refinement or quality restrictions:

- A variationally consistent mass matrix cannot be used. A consistent mass matrix violates the discrete form of the maximum/minimum principle, which can lead to violations of the reactive-definite property. A diagonally lumped mass matrix must be used. The particular approach to mass lumping can also introduce artifacts in the computed solution that depend on the connectivity of a mesh. A mass lumping that uses the control volume seems to eliminate these artifacts.

- If elements do not meet geometric constraints, nodes should be decoupled at the element level. This creates an inconsistent discretization which is no longer guaranteed to converge via h-refinement. For meshes consisting of multidimensional simplex elements (triangles and tetrahedra), no element can contain an interior obtuse angle. The finite volume geometric constraints (e.g. Delaunay triangulation in 2D) are not this restrictive. Furthermore, one of the major benefits of FEM discretizations, optimal convergence rates using higher order polynomials within an element, is also not recommended because it leads to more restrictive geometric constraints.

It should be reiterated that the above discretization requirements only apply to arbitrary mesh that is not subject to refinement or quality restrictions. An alternative to the use of those requirements would have been to develop refinement criteria which would guarantee that the reactive-definite property could be maintained, even with a consistent mass matrix and elements that do not meet geometric constraints. Because this research was not oriented toward mesh generation, this alternative was not pursued.

The techniques used to convert the information model into an executable representation are discussed in Chapter 5. The implementation of the spatial discretization is similar to many other nonlinear finite element codes in that a routine consumes elements and produces element-based residual vectors and tangent matrices. One major difference is that these vectors and matrices are only the portions associated with a single scalar equation in a system and the derivative of the discretized scalar equation taken with respect to a single field. These "sub"-element residual and tangents are

efficiently assembled into element residuals and tangents consider coupling among all of the terms in the equations in a region.

In addition to evaluation of the discretization, the scalar quantities used in a model also need to be converted into an executable representation. Rather than attempting to generate and externally compile code to be linked with the simulator, the evaluation is done in an interpreted fashion. Scalar quantities are represented by expression trees. To achieve efficiency, the following optimizations were implemented:

- Numerical operations work on blocks of data. This minimized interpretation overhead and increases the amount of instruction-level parallelism available to a compiler.

- In addition to element-based evaluation, nodal evaluation was also implemented. This capability increases the size of blocks that can be operated on and permits sharing of evaluated quantities among neighboring elements.

- Duplicated evaluation is eliminated by detecting identical subtrees in all expression trees in a system in region. The detection uses associative rules at evaluation nodes, where possible, but does not transform the expression tree into equivalent trees that may match as duplicates.

- Because of the nodal lumping of both the mass matrix and reactive terms, optimal element sparsity should be used for element tangent matrices for systems of equations in regions. This results in both a space savings for the storage of the global tangent matrix as well as a time savings for the matrix-vector multiply used in most iterative linear solution kernels.

Performance benchmarking shows that an implementation of interpretive evaluation can approach the performance of a hand-coded implementation.

In addition to its value, a scalar quantity must also be able to analytically differentiate itself with respect to a specified native field variable. With interpreted evaluation implemented using a expression tree, this is a fairly simple extension. In addition to value computation, each node also implements a derivative routine that propagates the node's contribution to the derivative using the chain rule.

In addition to the above technical contributions, practical aspects related to the overall design of simulation software were also investigated in Chapter 4. To ensure extensibility in a design, one must avoid basing the design on abstractions and assumptions that are likely to be invalidated during the lifetime of the software. As a specific example of limiting extensibility by choosing abstractions with a finite lifetime, the diffusion model abstractions in SUPREM-IV, which are based on a fully-coupled diffusion formulation, were discussed.

The growing complexity of CAD simulation software has also highlighted the need to pay attention to software integration in a design. Generic interfaces developed for ALAMODE to facilitate integration of external linear solution packages and access to semiconductor wafer mesh and field data were presented.

Chapter 6 discusses several examples of diffusion models that have been implemented and simulated in ALAMODE. The first model is a reduced system, which specifies a single equation for each impurity. Pairing reactions of impurities with native defects are assumed to have reached equilibrium as have the reactions involving interactions between native defects. The overall effect of the equation reductions is a diffusivity that depends on the local Fermi level, which in turn depends on the net active carrier concentration. The implementation of this example demonstrates the several key capabilities of ALAMODE, including: automatic differentiation (of the diffusivities), support for different systems of equations in different regions, and global coupling through an interface segregation flux.

The second model is a fully kinetic description of phosphorus diffusion which considers five species: mobile silicon interstitial and vacancies, mobile pairs of phosphorus with interstitials and vacancies, and unpaired immobile phosphorus. Over most of the device structure, the pairing reactions are in equilibrium, but there is a small region where the nonequilibrium in the reactions leads to the development of the characteristic kink and tail in the phosphorus profile. The pairing and recombination reactions are very strong and the diffusivities vary over several orders of magnitude. This results in a stiff system of equations to be integrated. The TR/BDF2 integration technique is able to efficiently integrate this system with time-steps controlled by estimates of the temporal error.

A more significant problem is that the linearized system of equations is poorly conditioned, especially with large time-steps. This poses a challenge for efficient multi-dimension simulation where iterative linear solution techniques may be required because of prohibitive memory requirements of direct factorization. Although somewhat expensive, the zero-fill incomplete LU preconditioned used by ALAMODE has proven to be very effective in providing a robust linear solution.

The third model is representative of the complexity of models used in current research into the mechanisms behind transient enhanced diffusion during quick low temperature anneals. This model provides kinetic treatment of the growth and dissolution of individually sized {311} defects. Such treatment is needed to properly model the effects of Ostwald ripening in which the average size of a defect increases with the anneal time. Because the interstitial release rate from {311} defects depends on the binding energy which depends on the defect size, properly modeling the evolution of the distribution of defect sizes is critical to predict the amount of TED for anneals that end before the extended defects completely dissolve.

7.2 Recommendations for Future Work

With respect to the development and improvement of rapid-prototyping simulation software, there are three main areas where there is room for improvement. The first area is the improvement of grid generation and adaptation. The second area is improving the numerical techniques used for spatial and temporal discretization as well as linear solution. The third area is to develop and support dial-an-operator based paradigms that permit implementation of PDE-based physical models drawn from other domains, such as the device equations.

Although ALAMODE has a versatile interface for plugging in external meshing tools, the present set of mesh utilities that have been integrated do not provide sufficient robust capabilities for full-scale process simulation. Features that are lacking include: generation of initial meshes from geometric representations (especially in 3D), the ability to add and remove interpolated fields at any time, persistent storage of the complete grid, field, and any additional structure (e.g. a quadtree or octtree)

needed to recover the state of a simulation, spatial adaptation driven by simulator-provided error measures, and robust handling of moving interfaces, such as those encountered during oxidation simulation.

The drawbacks of the spatial discretization were documented in Chapter 3. Other discretizations should be implemented and analyzed. One promising spatial discretization is an exponentially weighted finite element discretization developed by Mijalković [36]. His preliminary results show much better accuracy than can be achieved with linear finite element or finite volume discretizations. It also appears that geometric constraints on elements are field-dependent and tend to be more relaxed than for the linear discretization, but still require inconsistent nodal decoupling to fix. The form of the field-dependent geometric constraints have not yet been developed. The discretization also assumes a constant diffusivity over an element, leaving open the issue of what is the optimal measure of the effective diffusivity when it is not constant on an element. Finally, a formal error analysis needs to be derived, particularly for the case where this spatial discretization is used with nodally lumped reactive terms.

Although TR/BDF2 has performed well for temporal integration, semi-implicit approaches, such as the semi-implicit Runge-Kutta [9] algorithms, have been praised for superior stability properties over TR/BDF2 [26]. Another potential benefit of these algorithms is that they do not require a nonlinear solution to integrate a time interval. The latter could mean significant performance improvements if only a single global assembly and linear solution is needed for each time step instead of the 3-9 assembly/solves that are typically needed for the nonlinear iterations to solve a TR/BDF2 interval.

The final research area on the numerics side is the development of inexpensive preconditioners for iterative linear solution. Although ALAMODE's use of ILU as a preconditioner has proven to be robust, it is an expensive preconditioner.

The dial-an-operator framework developed in ALAMODE has proven itself for representation and solution of equation-based models derived for semiconductor diffusion, but it does not have all of the capabilities to effectively describe and simulate PDEs from other modeling areas for semiconductor devices, such as solution of the

device equations to obtain electrical characteristics or modeling of mechanical stress during thermal processing. Applying the dial-an-operator approach described in this dissertation to these areas would require some fundamental changes to the underlying information model and subsequent implementation. The most significant changes would be: 1) diffusive and advective terms would need to be discretized simultaneously to provide for a stable discretization (upwinding) and 2) the assumption that fields are scalar quantities would need to be removed, allowing for treatment of vector and tensor quantities. The latter would require fairly significant changes to the code which implements expression evalution, but this just provides further evidence that extensibility requires design for extensibility in a given area.

Even with the additional design and implementation effort to develop a simulator supporting a dial-an-operator paradigm, adding these capabilities definitely reduces overall development effort. The most obvious benefit is that models can be developed and changed very quickly. Specification and implementation of models is greatly simplified compared to coding a model in a traditional programming language. Finally, the number of implementation errors are reduced because the tedious and error prone tasks of discretization and linearization are taken out of the hands of a model implementor through the reuse of discretized components and automatic differentiation. Any discretized PDE solver would benefit from the inclusion of these capabilities.

Appendix A

Advective Stabilization for Semiconductor Diffusion

None of the simulators in the SUPREM family have implemented any advective stabilization for the electric field flux. PEPPER, however, provided a upwinded form of a drift operator. The following simplified analysis examines the stability requirements for the semiconductor diffusion equation.

The canonical advective-diffusive equation in 1D is given by

$$u\phi_{,x} = \kappa\phi_{,xx} + f \tag{A.1}$$

where u is a divergence-free *wind*, ϕ is the quantity that is drifting and diffusing in the wind, κ is the diffusivity, and f is a source of ϕ. The stability criterion for an unstabilized spatial discretization is given by the constraint on the element Peclet number (α)

$$\alpha = \frac{|u|h}{2\kappa} < 1 \tag{A.2}$$

where h is the mesh spacing.

The semiconductor impurity diffusion equation is given by

$$C_{,t} = \nabla\cdot\left(D\nabla C \pm \frac{DC}{V_T}\nabla\psi\right) + G - R \tag{A.3}$$

Assuming constant diffusivities and no generation or recombination, the semiconductor impurity diffusion equation reduces to (in 1D):

$$\begin{aligned} C_{,t} &= DC_{,xx} \pm \frac{D}{V_T}\frac{\partial}{\partial x}\left(C\psi_{,x}\right) && \text{(A.4)} \\ &= DC_{,xx} \pm \frac{D}{V_T}\left(C\psi_{,xx} + C_{,x}\psi_{,x}\right) && \text{(A.5)} \end{aligned}$$

If quasi-neutrality is assumed ($\psi_{,xx} = 0$), the wind meets the divergence-free condition, and the semiconductor impurity diffusion equation becomes

$$C_{,t} = \frac{D}{V_T}\left(V_T C_{,xx} \pm \psi_{,x}C_{,x}\right) \tag{A.6}$$

The stability requirement for the semiconductor impurity diffusion equation can be written as

$$\alpha = \frac{|\psi_{,x}|h}{2V_T} < 1 \tag{A.7}$$

If ψ is linearly varying over elements, then $|\psi_{,x}| = \Delta V/h$, where $\Delta V = |V_{n+1} - V_n|$ is the magnitude of voltage difference across the element. Thus,

$$\Delta V < 2V_T = 2kT \tag{A.8}$$

where k is Botzmann's constant and T is the absolute processing temperature. Note that this constraint holds regardless whether the electric field was generated primarily by the charged portion of the impurity in this equation or by the combination of charged impurities in this and other equations in the system.

Assuming a semiconductor with a bandgap of $1V$, a process simulator would need at least 5 evenly spaced grid points to resolve a half bandgap transition at 700^oC and 10 grid points to resolve a full bandgap transition to maintain spatial stability. 700^oC

is the current low end of thermal processing conditions and where the electric field effect is more pronounced. At 700^oC, a half bandgap transition involves a change in active doping levels of approximately four orders of magnitude. The heuristics for adequate mesh density for accuracy in a simulation require at least three to five grid points for each decade change in concentration. Provided that these accuracy heuristics are met, no spatial stabilization would be needed.

There are, of course, exceptional conditions under which the accuracy heuristics cannot be met. The most common is when one is trying to simulate introduction of a dopant via predeposition or a similar process, such as abutting a highly doped material against a lightly doped material. In this case, the initial condition is discontinuous at the surface, and no amount of refinement can be used to meet the stability requirement.

Finally, it is important to note that advective stabilization is obtained by increasing the diffusivity for the diffusive term in the equation. The diffusivity is only increased locally where needed for stabilization in the mesh. This is not desirable because doing so introduces an artificial effect that leads to increased mesh sensitivity and could adversely affect calibration.

Bibliography

[1] Semiconductor Industry Association. The national techonology roadmap for semiconductors, 1997.

[2] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, 1998.

[3] J. Banaszek. Comparison of control volume and galerkin finite element methods for diffusion-type problems. *Numerical Heat Transfer, Part B*, 16:59–78, 1989.

[4] Randolph E. Bank, William M. Coughran, Jr., Wolfgang Fichtner, Eric H. Grosse, Donald J. Rose, and R. Kent Smith. Transient simulation of silicon semiconductor devices. *IEEE Transactions on Electron Devices*, ED-30(10):1992–2007, October 1985.

[5] Christian Bischof, Lucas Roh, and Andrew Mauer-Oats. ADIC: An extensible automatic differentiation tool for ANSI-C. Technical Report ANL/MCS-P626-1196, Argonne National Laboratory, 1996.

[6] Barry W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, May 1988.

[7] C. Bonafos, D. Mathiot, and A. Claverie. Ostwald ripening of end-of-range defects in silicon. *Journal of Applied Physics*, 83(6):3008–30017, March 1998.

[8] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings, Redwood City, CA, second edition, 1994.

[9] J.R̃. Cash. Semi-implicit runge-kutta procedures with error estimates for the numerical integration of stiff systems of ordinary differential equations. *Journal of the Association for Computing Machinery*, 23(3):455–460, July 1976.

[10] Goodwin Chin. *Circuit-oriented IC Technology Simulation—Information Models and Integration Frameworks*. PhD thesis, Stanford Univeristy, 1992.

[11] I. Christie and C. Hall. The maximum principle for bilinear elements. *International Journal for Numerical Methods in Engineering*, 20(3):549–553, March 1984.

[12] P. G. Ciarlet. Discrete maximum principle for finite difference operators. *Aequationes Math.*, 4:338–352, 1970.

[13] P. G. Ciarlet and P.-A. Raviart. Maximum principle and uniform convergence for the finite element method. *Computer Methods in Applied Mechanics and Engineering*, 2:17–31, 1973.

[14] George F. Corliss and Andreas Griewank. Operator overloading as an enabling technology for automatic differentiation. Technical Report ANL/MCS-P358-0493, Argonne National Laboratory, 1993.

[15] Jack J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee, 1999.

[16] Kevin Dowd. *High Performance Computing*. O'Reilly & Associates, Inc., Sebastapol, CA, 1993.

[17] Scott T. Dunham. Modeling the kinetics of dopant precipitation in silicon. *Journal of the Electrochemical Society*, 142:2395–2397, April 1995.

[18] R. B. Fair. Recent advances in implantation and diffusion modeling for the design and process control of bipolar ics. In H. R. Huff and T. Abe, editors, *Proceedings of the 3rd International Symposium on Silicon Materials Science and Technology*, page 968, May 1977.

[19] R. Gelinas, S. Doss, and K. Miller. The moving finite element method: Applications to general partial differential equations with multiple large gradients. *Journal of Computational Physics*, 40:202–249, 1981.

[20] M. D. Giles, D. S. Boning, G. R. Chin, W. C. Dietrich Jr. M. S. Karasick, M. E. Law, P. K. Mozumder, L. R. Nackman, V. T. Rajan, D. M. H. Walker, R. H. Wang, and A. S. Wong. Semiconductor wafer representation for tcad. *IEEE Transactions on Computer-Aided Design*, 13(1):82–95, January 1994.

[21] S. Graham, P. Kessler, and M. McKusick. An execution profiler for modular programs. *Software – Practice and Experience*, 13:671–685, 1983.

[22] Peter Griffin. private communication, November 1997.

[23] Stephen E. Hansen. *SUPREM-III User's Manual*. Stanford University, 8628 edition, August 1986.

[24] Alan C. Hindmarsh. LSODE and LSODI, two new initial value ordinary differential equation solvers. *ACM SIGNUM Newsletter*, 15(4):10–11, 1980.

[25] W. Höhn and H. D. Mittelmann. Some remarks on the discrete maximum-principle for finite elements of higher order. *Computing*, 27(2):145–154, 1981.

[26] M. E. Hosea and L. F. Shampine. Analysis and implementation of TR-BDF2. *Applied Numerical Mathematics*, 20:21–37, 1996.

[27] Thomas J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[28] Werner Jüngling. *ZOMBIE User's Manual (version 1.1)*. Technical University of Vienna, 1987.

[29] Werner Jüngling, Peter Pichler, Sigfried Selbeherr, Edgar Guerrero, and Hans W. Pötzl. Simulation of critical IC fabrication processes using advanced physical and numerical methods. *IEEE Transactions on Electron Devices*, ED-32(2):156–167, February 1985.

[30] Mark E. Law. *Two Dimensional Numerical Simulation of Impurity Diffusion in Silicon*. PhD thesis, Stanford Univeristy, 1988.

[31] Mark E. Law. *FLOODS/FLOOPS Manual*. University of Florida, March 1995.

[32] Mark E. Law, Conor S. Rafferty, and Robert W. Dutton. *SUPREM-IV Users' Manual*. Stanford University, December 1988.

[33] Minchang Liang and Mark E. Law. An object-oriented approach to device simulation—floods. *IEEE Transactions on Computer-Aided Design*, 13(10):1235–1240, October 1994.

[34] Klas Liljia. private communication, October 1998.

[35] M. J. Maron. *Numerical Analysis: A Practical Approach*. Macmillan Publishing Company, New York, NY, second edition, 1987.

[36] Slobodan Mijalković. Exponentially fitted discretization schemes for diffusion process simulation on coarse grids. *IEEE Transactions on Computer-Aided Design*, 15(5):484–492, May 1986.

[37] B. J. Mulvaney, W. B. Richardson, and Timothy L. Crandle. PEPPER—a process simulator for VLSI. *IEEE Transactions on Computer-Aided Design*, 8(4):336–349, April 1989.

[38] Brian J. Mulvaney, Walter B. Ricardson, Greg Siebers, and Tim Crandle. PEPPER 1.2 user's manual. Technical Report CAD-239-90 (Q), Microelectronics and Computer Technology Corporation, 1989.

[39] P. Naur and B. Randell (eds.). Software engineering: A report on a conference sponsored by the nato science committee. Technical report, NATO, 1969.

[40] Marius Orlowski. Challenges for process modeling and simulation in the 90's–an industrial perspective, 1991. Invited talk at SISDEP '91.

[41] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, 1994.

[42] Anand L. Pardhanani and Graham F. Carey. Time-integration and iterative techniques for semiconductor diffusion modeling. *IEEE Journal of Technology Computer Aided Design*, 1998. SISPAD '97 Special Issue at `http://www.ieee.org/journal/tcad/`.

[43] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, New York, NY, second edition, 1987.

[44] C.S̃. Rafferty, G.H̃. Gilmer, M. Jaraiz, D. Eaglesham, and H.-J. Gossmann. Simulation of cluster evaporation and transient enhanced diffusion in silicon. *Applied Physics Letters*, 68(17):2395–2397, April 1996.

[45] Conor Rafferty. Progress in predicting transient diffusion. In *1997 International Conference on Simulation of Semiconductor Processes and Devices*, pages 1–4, Cambridge, MA, September 1997.

[46] Louis B. Rall. *Automatic differentiation : techniques and applications*. Springer-Verlag, New York, NY, 1981.

[47] Ernst Rank, Casimir Katz, and Heinrich Werner. On the importance of the discrete maximum principle in transient analysis using finite element methods. *International Journal for Numerical Methods in Engineering*, 19(12):1771–1782, December 1983.

[48] W. B. Richardson, G. F. Carey, and B. J. Mulvaney. Modeling phosphorus diffusion in three dimensions. *IEEE Transactions on Computer-Aided Design*, 11(4):487–496, 1992.

[49] W. B. Richardson and B. J. Mulvaney. Plateau and kink in p profiles diffused into si: A result of strong bimolecular recombination? *Applied Physics Letters*, 53(20):1917–1919, November 1988.

[50] Zakir Hussain Sahul. *Grid and Geometry Servers for Semiconductor Process Simulation*. PhD thesis, Stanford Univeristy, 1996.

[51] T. Saito, J. Xia, R. Kim, T. Aoki, H. Kobayashi, Y. Kamakura, and K. Taniguchi. Experiments and modeling of boron segregation to {311} defects and initial rapid enhanced diffusion induced by self-implantation in si. In *International Electron Devices Meeting Technical Digest*, pages 497–500, San Francisco, CA, December 1998.

[52] Sally Shlaer and Stephen J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[53] I. Sommerville. *Software Engineering*. Addison-Wesley, Wokingham, England, third edition, 1989.

[54] Stanford University. *SUPREM-IV.GS: Two Dimensional Process Simulation for Silicon and Gallium Arsenide*, 1993.

[55] Gilbert Strang and G. J. Fix. *An Analysis of the Finite Element Method*. Prentice Hall, Englewood Cliffs, NJ, 1973.

[56] Robert L. Taylor. *FEAP–A finite element analysis program: Version 6.4 Programmer Manual*. University of California at Berkeley, May 1998.

[57] Hal R. Yeager and Robert W. Dutton. An approach to solving multiparticle diffusion exhibiting nonlinear stiff coupling. *IEEE Transactions on Electron Devices*, ED-32(10):1964–1976, October 1985.

[58] Erich Zauderer. *Partial Differential Equations of Applied Mathematics*. John Wiley & Sons, New York, second edition, 1989.