AN ABSTRACT OF THE THESIS OF

Carl R. Huster for the degree of Masters of Science in
Electrical & Computer Engineering presented on July 29, 1992.

Title: A Parallel/Vector Monte Carlo MESFET Model for Shared Memory Machines

Abstract approved: Redacted for Privacy

S. M. Goodnick

The parallelization and vectorization of Monte Carlo algorithms for modelling charge transport in semiconductor devices are considered. The standard ensemble Monte Carlo simulation of a three parabolic band model for GaAs is first presented as partial verification of the simulation. The model includes scattering due to acoustic, polar-optical and intervalley phonons. This ensemble simulation is extended to a full device simulation by the addition of real-space positions, and solution for the electrostatic potential from the charge density distribution using Poisson's equation. Poisson's equation was solved using the cloud-in-cell scheme for charge assignment, finite differences for spatial discretization, and simultaneous over-relaxation for solution. The particle movement (acceleration and scattering) and the solution of Poisson's are both separately parallelized. The parallelization techniques used in both parts are based on the use of semaphores for the protection of shared resources and processor synchronization. The speed increase results for parallelization with and without vectorization on the Ardent Titan II are presented. The results show saturation due to memory access limitations at a speed increase of approximately 3.3 times the serial case when four processors are used. Vectorization alone provides a speed increase of approximately 1.6 times when compared with the nonvectorized serial case. It is concluded that the speed increase achieved with the Titan II is limited by memory access considerations and that this limitation is likely to plague shared memory machines for the forseeable future. For the program presented here, vectorization is concluded to provide a better speed increase

per day of development time than parallelization. However, when vectorization is used in conjunction with parallelization, the speed increase due to vectorization is negligible.

A Parallel/Vector Monte Carlo MESFET Model for Shared Memory Machines

by

Carl R. Huster

A Thesis
submitted to
Oregon State University

in partial fulfillment of the
requirements for the degree of

Masters of Science

Completed July 29, 1992
Commencement June 1993.

APPROVED:

Redacted for Privacy

Associate Professor of Electrical & Computer Engineering in charge of major

Redacted for Privacy

Head of Department of Electrical & Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis presented July 29, 1992

Typed by Carl R. Huster

ACKNOWLEDGEMENTS

There is a long list of individuals who deserve to be acknowleged here:

Dr. Stephen Goodnick – for guidance, moral support and financial support. Thanks for having me as a graduate student.

Dr. Udaya Ranawake – for his colaboration on this research.

Jessica Wang – for help with preparing figures, proofreading and lot of delicious meals.

Dr. Keya Bhattacharyya – for having patience with my endless stream of questions.

Dr. Jenifer Lary – for contributing her EE519 project as a starting point for this research.

Andrew Churchill and Thach Nguyen – for being great friends and helping keep my spirits up during the past two years.

George and Lee Huster (Alias Mom & Dad) – for the generosity and foresight to give me a 4K Radio Shack Color Computer 12 long years ago; for teaching me the value of education; and for the love, support and encouragement they have given.

Don & Alan Huster – for allowing me to learn from their successes, failures, and dreams; you two are the best brothers a guy could ask for.

The National Science Foundation – for support under grant number ECS-8821107.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Design, fabrication and testing of new solid state devices are very time consuming and expensive processes. It is therefore desirable to be able to simulate a device before the cost associated with fabrication and testing is incurred. Such simulations based on the drift-diffusion (DD) model have been in use for a number of years. The DD model operates by solving the continuity equations coupled with the drift-diffusion equations. Implicit in the DD approach is the assumption of a local relationship between the electric field and the carrier velocity. Hence, the DD model ignores the time required for the carriers to accelerate to their steady state velocity. This model works acceptably well for devices where the carrier transport properties are dominated by the steady state velocity-field characteristics. This property is normally true for devices with channel lengths greater than approximately 1μm. Present fabrication technology allows quantity production of integrated circuits containing devices with 0.7μm channel lengths. If simulation is to remain an effective tool, other techniques must be resorted to. The Monte Carlo (MC) particle method of modelling carrier transport is often resorted to when the DD model fails. In this method, the random number generation is used to simulate the random walk of carriers in the crystal. The MC method has the advantage that transient, nonstationary properties of the carriers are modelled, at least semiclassically. The primary disadvantage of MC is that it has much greater computational requirements than the DD model. One of the most important causes of the large computational requirements is the fact that the error scales as $N^{-\frac{1}{2}}$, where N is the number of particles simulated. Due to computer limitations, a typical number particles for a simulation is 10^5, which seems large until it is compared with the approximately 10^8 particles that actually exist.

Some developments in computer architecture in recent years hold promise to be able to more effectively meet the computational requirements of MC simulations. These architectures fall into two general categories: Single Instruction Multiple Data stream (SIMD), or Multiple Instruction Multiple Data stream (MIMD). The SIMD architecture is best typified by vector processors. Vector processors are typically capable of rapidly performing a variety of simple mathematical operations on one dimensional arrays of numbers, i.e. vectors. The variety of MIMD machines makes it impossible to present any one example that adequately represents the majority of such machines. Generally such machines can be put into two subcategories, shared memory machines and distributed memory machines. Shared memory machines are distinguished by having the memory accessible by all the processors. Distributed memory machines are the converse, the memory is associated locally with each processor.

This research considers the effectiveness of two ways of meeting these requirements. First, a MC device simulation was developed to exploit the parallel processing capability of a multi-processor shared memory computer, the Ardent Titan II. The Titan is described in detail in section 3.2. Second, the simulation was extended to exploit the vector processor associated with each of the four parallel processors. The parallelization was implemented by dividing certain tasks into independent parts which can be arbitrarily assigned to any of the processors. The exact techniques used are discussed in detail in section 3.3. Vectorization was accomplished by identifying loops with complicated expressions and reducing them to a series of loops simple enough for the vector processor to deal with.

The results, presented in sections 4.4 and 4.5, show saturation that is characteristic of shared memory computers. Both the vectorized and the non-vectorized case show a speed increase of about 3.3 times that for the non-vectorized single processor case. This saturation is due to memory access limitations. The saturation, however, gives a distorted view of what vectorization can accomplish. For the single processor case, vectorization provided a speed increase of 1.6 times.

Chapter 2

Monte Carlo Simulation of Semiconductor Devices

2.1 Overview

The drift-diffusion and Monte Carlo (MC) techniques introduced in chapter 1 represent different approaches for solving the more general Boltzmann Transport Equation (BTE). The BTE is a kinetic equation for the single particle carrier probability distribution function which accounts for all possible mechanisms by which the function may change. The drift diffusion approach represents the solution to the BTE through the first two moments of this equation, which gives the so called drift-diffusion equation commonly used in analytical semiconductor theory. For the MC method, carrier transport is modelled by subjecting each particle to a random walk process. The random walks consist of two parts, free-flight interrupted by instantaneous scattering events. During free-flight, the changes in the particle position and wavevector are governed by electric and magnetic fields as well as internal forces due to spatial inhomogenaities. Scattering events are assumed to instantaneously change the wavevector of the particle, but not the position. Scattering occurs due to random imperfections in the crystalline lattice such as ionized impurities, vibrations (phonons), or othe carriers. These rates at which the various scattering events occur are, in general, functions of energy. The scattering rates are the topic of section 2.2.2. For device simulation, both solution techniques consist of two main elements, solution of the transport equation and solution of the electrostatic potential over the spatial domain.

When the MC method is applied to a homogeneous bulk material, it is referred to as a $\vec{k}$-space or an ensemble simulation. In such a simulation, it is not necessary to solve for the electric field since every particle will be subject to the same average accelerating field. In an actual device simulation, it is necessary to solve for the electric field at every point in the device. Assuming that a quasi-static approximation is valid, the electric field is related to the gradient of the electrostatic potential which is the solution to Poisson's equation for the known charge density distribution. There are at least two general schemes for discretizing Poisson's equation on to a mesh; finite differences and finite elements. Both methods produce a set of simultaneous linear equations which may be solved by a variety of techniques discussed in section 2.3. The electric field computed from the potential is used to accelerate the particles during one time step when they are in free-flight. After all the particles have completed a particular time step, the new charge configuration due to the movement of the particles is assigned to the mesh. Poisson's equation is then updated in order to calculate the new electric field over the next time step. The general flow of a typical MC device simulation based on this algorithm is shown in figure 2.1.

2.2 Particle Movement

Figure 2.2 is a flow chart of the essential elements of particle movement. The subsections which follow deal with various aspects of the process shown in figure 2.2. Subsection 2.2.1 begins by introducing the method of selecting the random free-flight durations. The selection of these times is the basis of the Monte Carlo method. The free-flight durations are determined from the scattering rates which are presented in subsection 2.2.2. Subsection 2.2.3 presents how free-flight and the various scattering mechanisms are performed. In subsection 2.2.4, the real space considerations associated with device simulation are discussed. These considerations can be generally described as a specification of what happens to a particle when it attempts to exit the simulated region. Subsections 2.2.5 and 2.2.6 present two interesting, but less essential portions of the transport portion of the simulation. Subsection 2.2.5

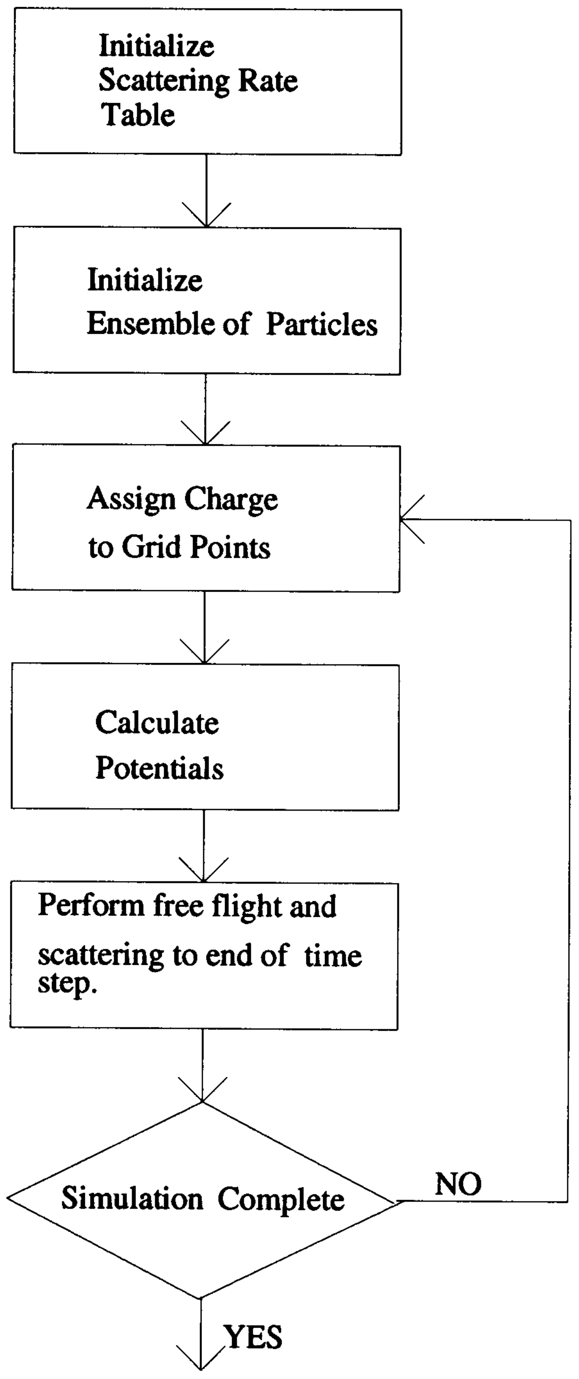


Figure 2.1: Basic flow chart of a Monte Carlo device simulation.

details the initial $\vec{k}$-space and real space distributions of particles. Subsection 2.2.6 presents the random number generator used throughout the simulation.

2.2.1 Free Flight Times

One of the most important parts of the MC method is the determination of when scattering events should occur, or equivalently how long the free flights should last. The flight time is a function of the scattering rates to which the particle is subject (see section 2.2.2). As will be shown in section 2.2.2, these scattering rates are functions of energy in general. This fact is accounted for by expressing the probability of scattering in an interval dt about t as a general function of the wavevector $\vec{k}(t)$,

$$P[\vec{k}(t)]dt. \tag{2.1}$$

Now, a large number of particles with identical wavevectors beginning free flight at $t = 0$ are considered. The rate of initial scatterings will be proportional to the number of surviving particles and the current scattering probability,

$$\frac{\partial n(t)}{\partial t} = -P[\vec{k}(t)]n(t), \tag{2.2}$$

where $n(t)$ is the number of particles surviving at time t. The general solution to equation 2.2 is

$$n(t) = K \exp\left\{-\int_0^t P[\vec{k}(t')]dt'\right\}, \tag{2.3}$$

where K is a constant. K is easily determined to be $n(0)$. The number of particles surviving at time t is simply

$$n(t) = n(0) \exp\left\{-\int_0^t P[\vec{k}(t')]dt'\right\}. \tag{2.4}$$

The probability of a single particle surviving to time t is found by normalizing equation 2.4 with respect to $n(0)$,

$$Q(t) = \frac{n(t)}{n(0)} = \exp\left\{-\int_0^t P[\vec{k}(t')]dt'\right\}. \tag{2.5}$$

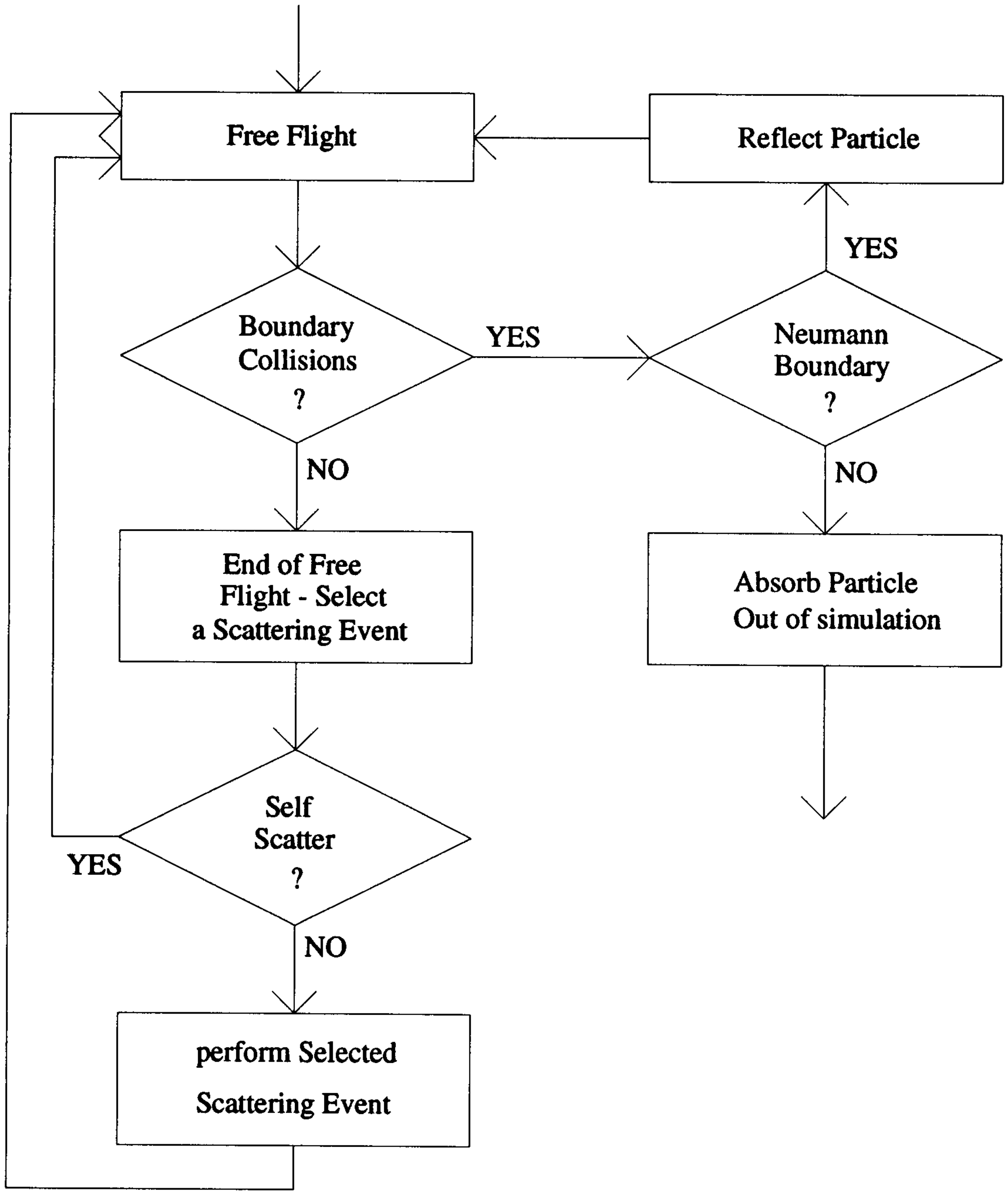

Free Flight
Reflect Particle
YES
Boundary Collisions ?
YES
Neumann Boundary ?
NO
NO
End of Free Flight - Select a Scattering Event
Absorb Particle Out of simulation
Self Scatter ?
YES
NO
perform Selected Scattering Event

Figure 2.2: Flow chart of particle movement.

The probability of the first scattering event since $t = 0$ happening during an interval dt about t is simply the joint probability of the particle surviving to time t and scattering at time t,

$$R(t)dt = Q(t)P[\vec{k}(t)]dt \tag{2.6}$$

or equivalently,

$$R(t) = \exp\left\{-\int_0^t P[\vec{k}(t')]dt'\right\}P[\vec{k}(t)]dt. \tag{2.7}$$

Random flight times sampled from the nonuniform probability distribution $R(t)$ may be obtained from a uniform random variable r in the range [0,1) by the direct technique given in [3]. This procedure gives

$$r = \frac{\int_0^{t''} R(t')dt'}{\int_0^{\infty} R(t')dt'}, \tag{2.8}$$

or equivalently,

$$r = \frac{\int_0^{t_r} \exp\left\{-\int_0^{t'} P[\vec{k}(t'')]dt''\right\}P[\vec{k}(t')]dt'}{\int_0^{\infty} \exp\left\{-\int_0^{t'} P[\vec{k}(t'')]dt''\right\}P[\vec{k}(t')]dt'}, \tag{2.9}$$

where t_r is a random number satisfying the nonuniform distribution given in equation 2.6 and r is a uniform random variable in the range $[0, 1)$. Since $P[\vec{k}(t)]$ as a function of time is not known apriori, 2.9 is not easily evaluated. There is, however, a remedy for the complexity of equation 2.9. If the total scattering rate, $P[\vec{k}]$, is a constant, say Γ, then equation 2.9 reduces to

$$r = 1 - \exp(-\Gamma t_r). \tag{2.10}$$

This can easily be solved for the flight time in terms of a uniform random variable,

$$t_r = -\Gamma^{-1}\ln(r). \tag{2.11}$$

Now, the problem that remains is how to force $P[\vec{k}(t)]$ to be a constant without altering the physics. Suppose that $P[\vec{k}(t)]$ is written as

$$P[\vec{k}(t)] = P_{Real}[\vec{k}(t)] + P_{Self}[\vec{k}(t)], \tag{2.12}$$

where the physical scattering mechanisms are accounted for in $P_{Real}[\vec{k}(t)]$. $P_{Self}[\vec{k}(t)]$ represents the scattering rate for an artificial scattering mechanism defined by

$$\vec{k}_{final} = \vec{k}_{initial}. \tag{2.13}$$

This mechanism, known as self scattering, does not affect the trajectory of the particles and hence the physics of the simulation. Thus, $P_{Self}[\vec{k}(t)]$ can be arbitrarily defined. The constant nature of $P[\vec{k}(t)]$ is achieved by defining

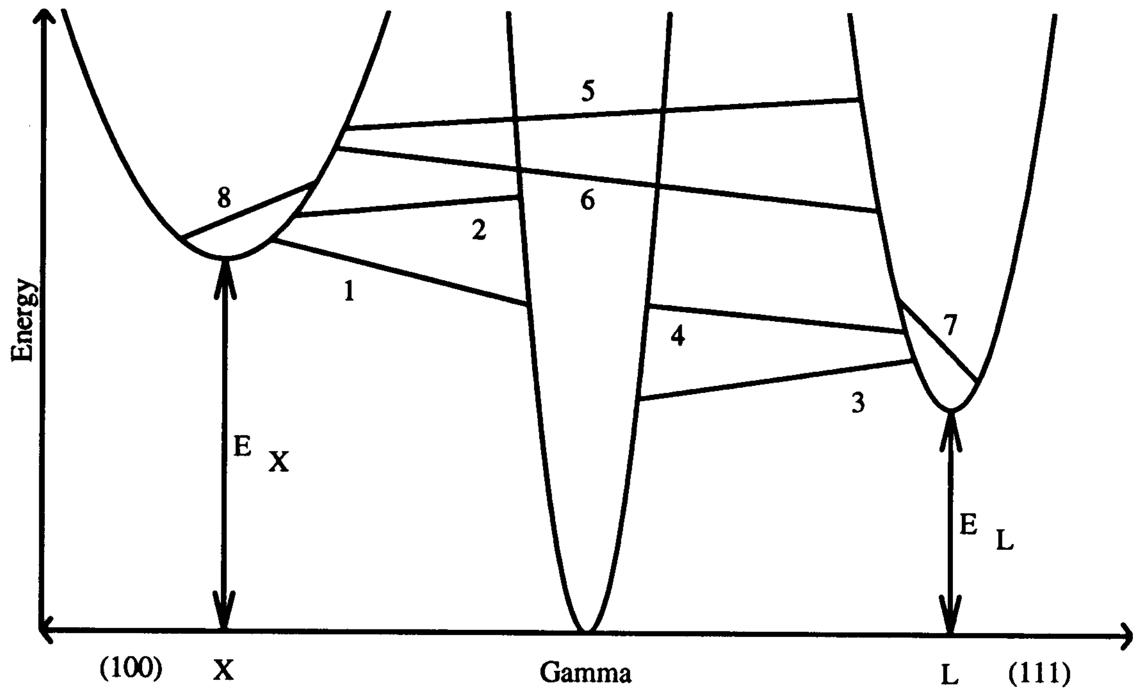
$$P_{Self}[\vec{k}(t)] = \Gamma - P_{Real}[\vec{k}(t)]. \tag{2.14}$$

The result is that equation 2.11 is the expression used for generating flight times. The cost of this simplification is that self scattering may account for over 90% of the scattering events, which can represent a significant amount of computer time. A technique that may help reduce this cost by the use of a piecewise constant Γ is presented by Yorston in [4].

2.2.2 Scattering Rate Calculations

As a carrier moves through the crystalline lattice of the semiconductor, it experiences a periodic potential due to the lattice, and a random potential due to imperfections in the lattice. The periodic potential gives rise to the renormalized mass. The lattice imperfections can be due to other carriers, impurities, and/or lattice vibrations (phonons). In the simulation presented here, only scattering mechanisms due to lattice vibrations are considered for simplicity. The types of mechanisms included are acoustic phonon scattering, polar optical phonon scattering, and intervalley scattering. Briefly, GaAs has three conduction band minima (valleys) of importance which are centered at the Γ, L, and X points in $\vec{k}$-space, shown schematically in figure 2.3. The scattering mechanisms included for each valley are listed in table 2.1, and discussed briefly below. The interested reader can refer to chapter 2 of [5] for more details of the scattering rate derivations.

In the simulation presented here, spherical parabolic bands were assumed.



Number	Descriptions	Number	Descriptions
1	Γ to X Intervalley Absorption	5	X to L Intervalley Absorption
	X to Γ Intervalley Emission		L to X Intervalley Emission
2	Γ to X Intervalley Emission	6	X to L Intervalley Absorption
	X to Γ Intervalley Absorption		L to X Intervalley Emission
3	Γ to L Intervalley Absorption	7	L to L Intervalley Absorption
	L to Γ Intervalley Emission		L to L Intervalley Emission
4	Γ to L Intervalley Absorption	8	X to X Intervalley Absorption
	L to Γ Intervalley Emission		X to X Intervalley Emission

Figure 2.3: GaAs intervalley scattering mechanisms.

Γ Valley	Acoustic Phonon
	Polar Optical Phonon Absorption
	Polar Optical Phonon Emission
	Γ to L Intervalley Absorption
	Γ to L Intervalley Emission
	Γ to X Intervalley Absorption
	Γ to X Intervalley Emission
L Valley	Acoustic Phonon
	Polar Optical Phonon Absorption
	Polar Optical Phonon Emission
	L to Γ Intervalley Absorption
	L to Γ Intervalley Emission
	L to L Intervalley Absorption
	L to L Intervalley Emission
	L to X Intervalley Absorption
	L to X Intervalley Emission
X Valley	Acoustic Phonon
	Polar Optical Phonon Absorption
	Polar Optical Phonon Emission
	X to Γ Intervalley Absorption
	X to Γ Intervalley Emission
	X to L Intervalley Absorption
	X to L Intervalley Emission
	X to X Intervalley Absorption
	X to X Intervalley Emission

Table 2.1: Scattering mechanisms included in the simulation.

This assumption is written as

$$E = \frac{\hbar^2(\vec{k}\cdot\vec{k})}{2m^*}, \tag{2.15}$$

where m^* is given by

$$\frac{1}{m^*} = \frac{1}{\hbar^2}\frac{\partial^2 E(|\vec{k}|)}{\partial|\vec{k}|^2}. \tag{2.16}$$

The particle velocity is defined by

$$\vec{v} = \frac{1}{\hbar}\frac{dE(\vec{k})}{d\vec{k}}. \tag{2.17}$$

Since the relationship between E and $\vec{k}$ is assumed to be given by 2.15, equation 2.17 simplifys to

$$\vec{v} = \frac{\hbar\vec{k}}{m^*}. \tag{2.18}$$

There are two assumptions which are usually used in deriving the acoustic phonon scattering rates. The first assumption is that, at room temperature, it is an elastic isotropic process due to the small energy of acoustic phonons. That is to say, acoustic phonon scattering can randomly change the direction, but not the magnitude of the carrier wavevector. The second assumption, known as the equipartition assumption, is that the acoustic phonon energy is less than the thermal energy so that the phonon occupancy, N_q, may be written as

$$N_q \cong N_q + 1 \cong \frac{k_B T_L}{\hbar\omega_q}, \tag{2.19}$$

where $\hbar\omega_q$ is the acoustic phonon energy. With these two assumptions, the scattering rate due to acoustic phonons may be written as

$$\frac{1}{\tau_{AC}} = \frac{D_A^2 k_B T_L (2m^*)^{3/2}}{2\pi\hbar^4 v_s^2 \rho}E^{1/2}, \tag{2.20}$$

where D_A is the acoustic deformation potential, T_L is the lattice temperature, v_S is the sound velocity, and ρ is the mass density. Since the quantity D_A is somewhat subjective, it is worth noting that values of 7.0eV, 9.2eV, and 9.0eV were used for the Γ, L and X valleys, respectively [5].

Scattering by polar optical phonons is an important mechanism for compound semiconductors like GaAs. Polar optical scattering results from the fluctuating dipole moment of the cation-anion pairs. For the case of a phonon absorption, the scattering rate is given by

$$\frac{1}{\tau_{POPABS}} = \frac{q^2\omega_0\left(\frac{\kappa_0}{\kappa_\infty} - 1\right)}{2\pi\kappa_0\epsilon_0\hbar\sqrt{\frac{2E}{m^*}}}\left[N_0 \sinh^{-1}\left\{\sqrt{\frac{E}{\hbar\omega_0}}\right\}\right], \tag{2.21}$$

where κ_0 is the low frequency dielectric constant, κ_∞ is the high frequency dielectric constant, $\hbar\omega_0$ is the optical phonon energy and N_0 is the Bose-Einstein distribution given by

$$N_0 = \frac{1}{\exp\left\{\frac{\hbar\omega_0}{k_B T_L}\right\} - 1}. \tag{2.22}$$

For the case of a carrier emitting a phonon, the scattering rate is given by

$$\frac{1}{\tau_{POPEMS}} = \frac{q^2\omega_0\left(\frac{\kappa_0}{\kappa_\infty} - 1\right)}{2\pi\kappa_0\epsilon_0\hbar\sqrt{\frac{2E}{m^*}}}\left[(N_0 + 1) \sinh^{-1}\left\{\sqrt{\frac{E}{\hbar\omega_0}}\right\}\right], \tag{2.23}$$

where it is understood that the condition $E \geq \hbar\omega_0$ must be met for emission to be possible.

Intervalley scattering corresponds to absorption or emission of a phonon which scatters a carrier from one valley to another. This process may occur between equivalent minima of the same valley or non-equivalent minima of different valleys. As mentioned earlier in this section, for electrons in GaAs, there are three conduction band minima (valleys). The Γ point in the Brillouin zone is the central valley is the central valley and has the lowest base energy. The L and X valleys are 0.29eV and 0.48eV above the Γ valley, respectively, as shown in figure 2.3.

The intervalley scattering rates are given by

$$\frac{1}{\tau_{IV}} = \frac{\pi D_{if}^2 Z_f}{2\rho\omega_{if}}\left(N_i + \frac{1}{2} \mp \frac{1}{2}\right) g_C(E \pm \hbar\omega_{if} - \Delta E_{fi}), \tag{2.24}$$

where D_{if} is the intervalley deformation potential between the initial and final valleys, Z_f is the degeneracy of the final valley, $\hbar\omega_{if}$ is the energy of the intervalley phonon, ΔE_{fi} is the energy offset of the final valley relative to the initial valley, g_C

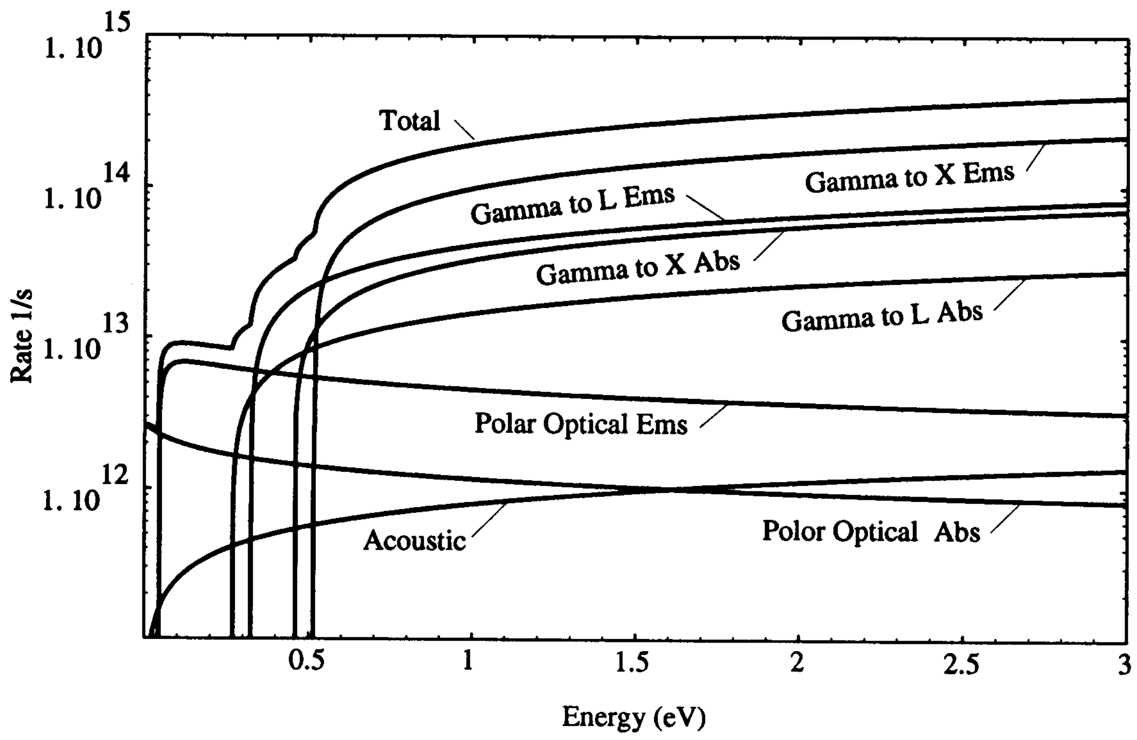


Figure 2.4: Γ valley scattering rates.

is the density of states for the final valley, and N_i is the number of phonons with energy $\hbar\omega_{if}$ again given by the Bose-Einstein factor,

$$N_i = \frac{1}{\exp\left\{\frac{\hbar\omega_{if}}{k_B T_L}\right\} - 1}. \tag{2.25}$$

As shown by the above scattering rate formulas, there are a large number of material parameters that are required. For the simulation presented here, the parameters for GaAs were taken from section 2.12 of [5]. The scattering rates calculated for the conduction band according to the rates presented above are shown in figures 2.4, 2.5 and 2.6.

2.2.3 Acceleration and Scattering

The flight of a particle can be interrupted by three events, a scattering event, a collision with a device boundary, or the end of a simulation time step. The three types of interruptions to the free-flight are shown in figure 2.7. Boundary collisions may represent an actual physical events depending on the type of boundary

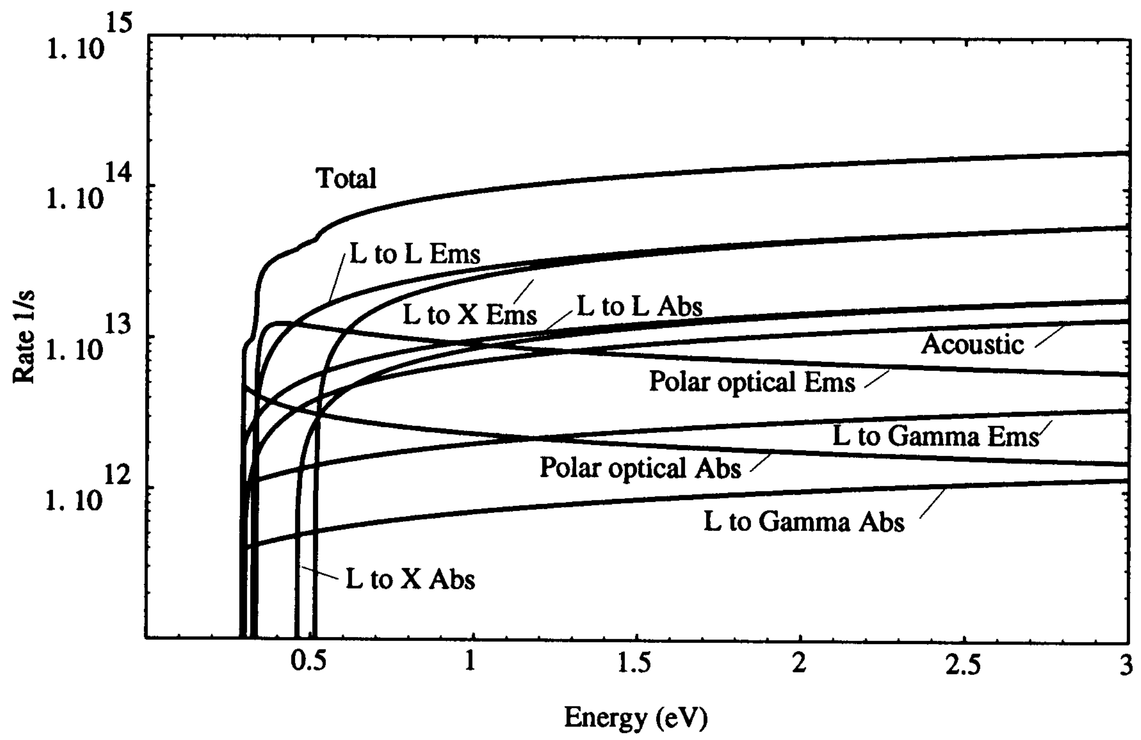


Figure 2.5: L valley scattering rates.

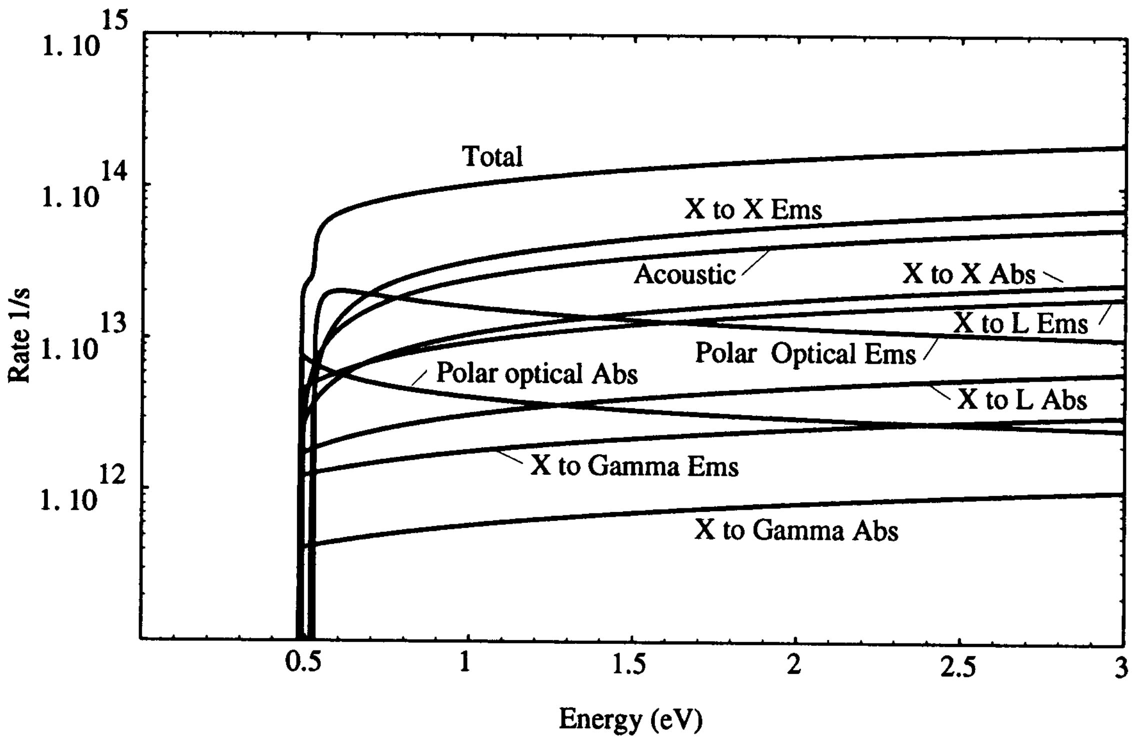


Figure 2.6: X valley scattering rates.

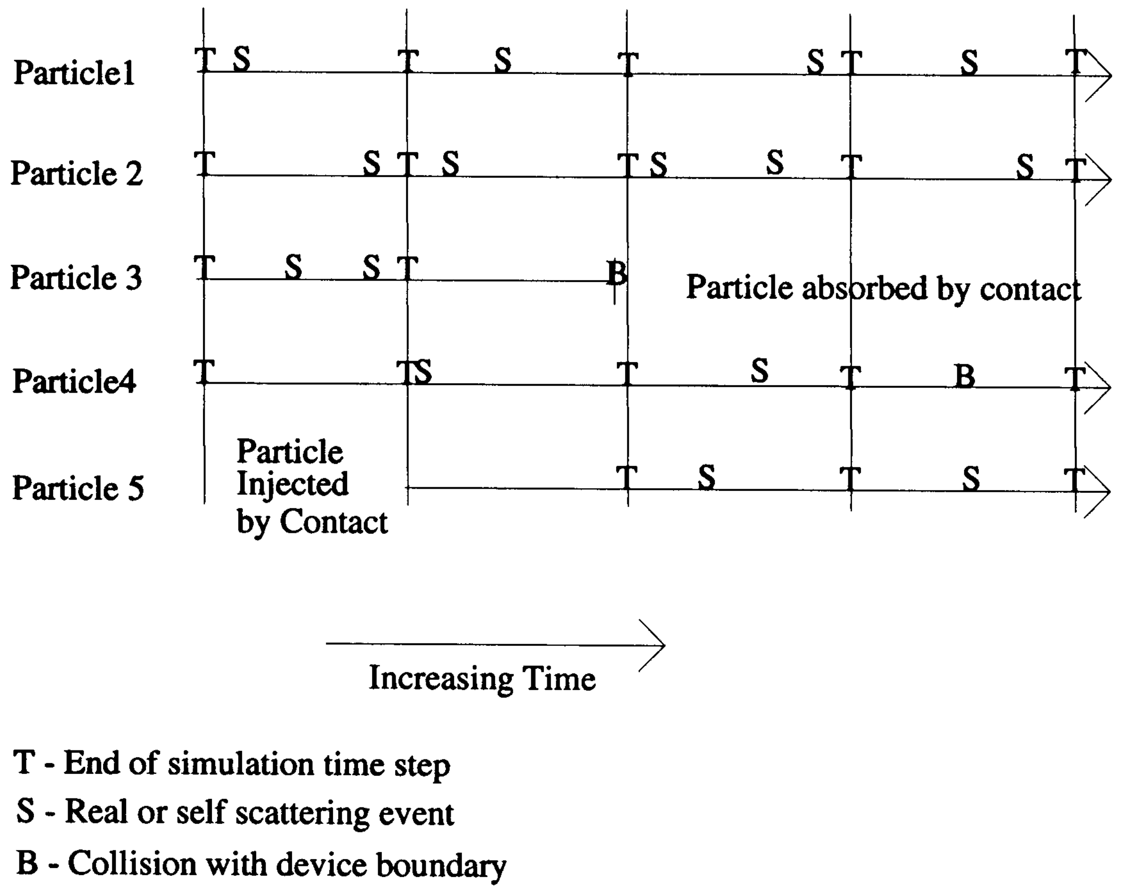

Particle1
Particle 2
Particle 3
Particle4
Particle 5
T
S
B
Particle absorbed by contact
Particle Injected by Contact
Increasing Time
T - End of simulation time step
S - Real or self scattering event
B - Collision with device boundary

Figure 2.7: Particle flights

encountered. Scattering was discussed in section 2.2.2. The simulation time step is nonphysical and is imposed to allow for the completion of certain necessary tasks. The most notable of these tasks is solving Poisson's equation in the device (discussed in section 2.3.2). If the simulation time step is chosen too large, then the simulation will no longer provide an accurate model. Specifically, the simulation may become unstable when too large of a time step is chosen [2]. The simulation time step used here was 0.01ps also as suggested in [2].

Two of the variables that must be tracked for each particle in the simulation are the wavevector and position. Assuming the particle is subject to a constant electric field, $\vec{\mathcal{E}}$, during the interval t_1 to t_2, and drawing on equation 2.15, the change in the particle's wavevector can be written as

$$\vec{k}(t_2) = \vec{k}(t_1) + \frac{\vec{\mathcal{E}}q(t_2 - t_1)}{\hbar}. \tag{2.26}$$

Now applying equation 2.18 and the acceleration therom to equation 2.26 give the solution for the particle position as

$$\vec{x}(t_2) = \vec{x}(t_1) + \frac{\hbar\vec{k}(t_1)(t_2 - t_1)}{m^*} + \frac{-\vec{\mathcal{E}}q(t_2 - t_1)^2}{2m^*}. \tag{2.27}$$

These equations govern the motion of the particles during the free-flight periods.

When a free-flight ends due to a scattering event, it is necessary to decide which scattering event occurs. At the end of the free-flight, the energy and valley for the particle are known. Associated with these quantities is a specific set of scattering probabilities. These probabilities are found by normalizing the scattering rates shown in figures 2.4, 2.5, and 2.6, relative to the maximum scattering rate, Γ. The type of scattering event terminating the free flight (including self-scattering) is determined by comparing a uniform random number in the range [0,1) with the normalized scattering rates.

Once a real scattering event is chosen, it is necessary to choose the final wavevector. This process involves selecting changes to all three polar coordinates of the wavevector, $\left|\vec{k}\right|$, θ, ϕ, relative to the initial wavevector according to the quantum mechanical scattering rate. The polar coordinate system is defined as shown in figure 2.8. Figure 2.8 also defines other angles relative to the fixed coordinate systems. The change in $\left|\vec{k}\right|$ is chosen to conserve both energy and momentum in the particle-phonon system. In scattering events where the energy of the particle changes, the new wave vector is calculated as

$$|\vec{k}| = \frac{\sqrt{2m^*(E \pm \Delta E)}}{\hbar}, \tag{2.28}$$

assuming parabolic bands. In the case of elastic scattering, ΔE is zero by definition. For the case of isotropic scattering mechanisms, the angles θ, and ϕ are chosen at random,

$$\phi = 2\pi r_1 \tag{2.29}$$

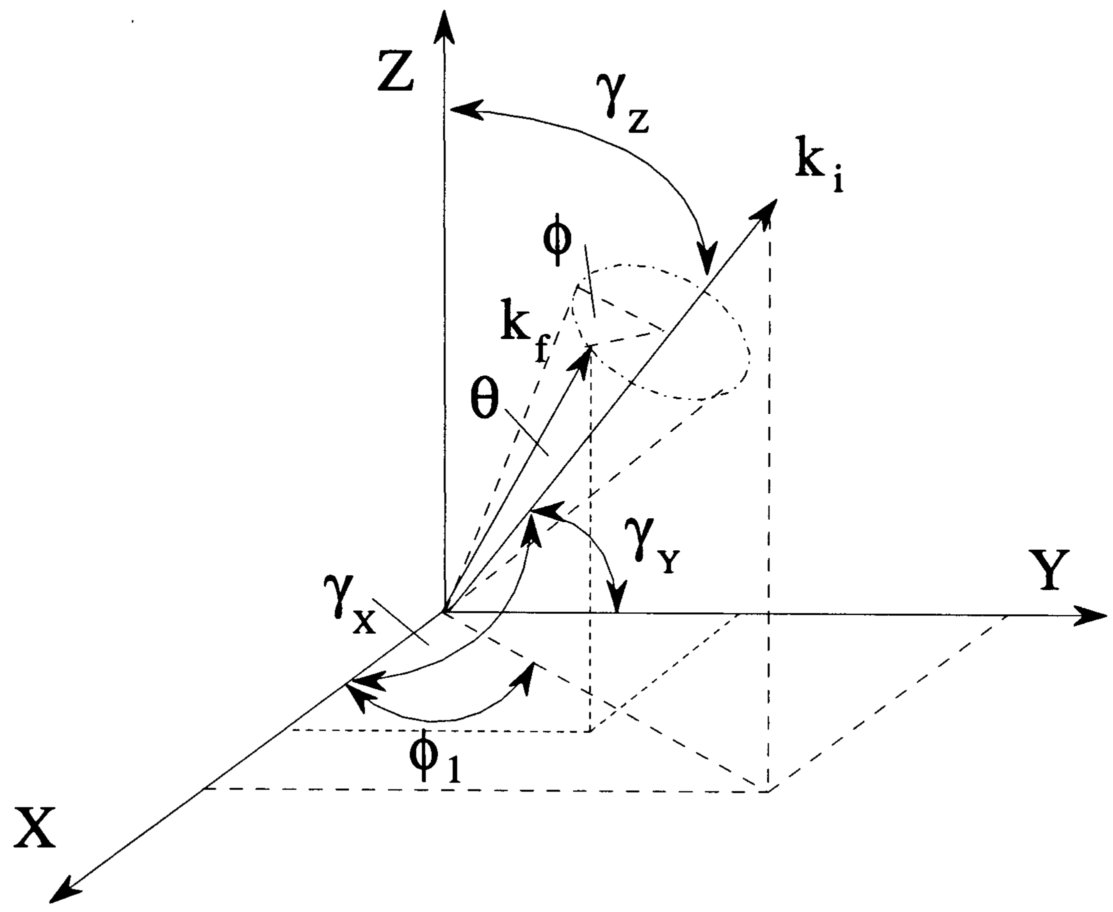


Figure 2.8: Polar coordinate system used for wavevectors, after [1]

and

$$\theta = \arccos(2r_2 - 1), \tag{2.30}$$

where r_1 and r_2 are uniform random numbers in the range [0,1).

In the case of anisotropic scattering such as that due to polar optical phonon scattering, the choice of ϕ is the same, but θ must be chosen according to

$$P(\theta)d\theta = \frac{\sin(\theta)d\theta}{2E \pm E_{POP} - \sqrt{E(E \pm E_{POP})}\cos(\theta)}, \tag{2.31}$$

which favors small angle scattering. Here E_{POP} is the optical phonon energy. For a derivation of 2.31, the reader can refer to chapter 2 of [3]. The rejection technique given in appendix B of [3] is used for selection of θ given equation 2.31. The choices

	Acoustic	Intervalley	Polar Optical Phonon
Elastic	Yes	No	No
Isotropic	Yes	Yes	No
P(ϕ)	$\frac{1}{2\pi}$	$\frac{1}{2\pi}$	$\frac{1}{2\pi}$
P(θ)	$\sin(\theta)$	$\sin(\theta)$	$\frac{\sin(\theta)}{2E \pm E_{POP} - \sqrt{E(E \pm E_{POP})}\cos(\theta)}$
$\lvert\vec{k}_2\rvert$	$\lvert\vec{k}_1\rvert$	$\frac{\sqrt{2m^*(E \pm \Delta E)}}{\hbar}$	$\frac{\sqrt{2m^*(E \pm \Delta E)}}{\hbar}$

Table 2.2: Properties used for selecting the wave vector after a scattering event.

of the components of $\vec{k}$ are summarized in table 2.2.

2.2.4 Contacts and Boundary Conditions

At every point on the boundary of a simulated device there exist conditions which govern the behavior of particles incident on that boundary. There are several types of boundary conditions that may exist in a given device. These include artificial internal boundaries with the bulk semiconductor, boundaries with an insulator, Ohmic contacts and Schottky barrier contacts.

Internal bulk boundary conditions are artificial and necessitated by the finite domain required in any numerical simulation involving a mesh. By definition, at such a boundary, there should be no effect to the device behavior in terms of particle flux or electric field. Thus such a boundary is usually assumed to be far removed from the area of interest. Although particles will routinely be incident on such a boundary, there should be no net flux of particles across the boundary. This condition is accomplished by introducing an artificial scattering mechanism which reflects the component of the momentum perpendicular to the boundary. The validity of this approach can be easily seen by considering the reflection as two separate events. The first is a particle with a wavevector (k_x, k_y, k_z) exiting the device. The second event is a particle with wavevector $(-k_x, k_y, k_z)$ entering the device. Hence there is no energy or particle transfer across the boundary.

In the simulation presented here, boundaries between the simulated device and insulating materials were treated identically to boundaries with the bulk regions. The MESFET simulated here is discussed in section 4.3 For such a device, boundaries with insulating materials exist only in the regions between the contacts on the top surface. Since the conducting channel in a MESFET does not abut this area, the treatment of these boundaries is not particularly important to the device simulation. However, in the case of a device such as a MOSFET where the entire conducting channel abuts such a boundary, considerably more care should be taken in how the boundary is modelled. For such a device, treatment might include scattering due to surface roughness and/or interface states.

A Schottky barrier contact presents an interesting problem to model. The model used in this simulation treats it as an absorbing, but noninjecting contact with a negative potential offset due to the barrier height. The justification for this treatment is quite simple. Since the barrier height is more than $20k_BT$ above the Fermi level in the metal, there are essentially no electrons with enough energy to surmount the barrier, except under strong forward bias conditions. The negative potential offset models the Schottky barrier for the electrons in the semiconductor. Although it is a rare event, should a carrier reach the Schottky contact, it is assumed to be absorbed out. This treatment is also convenient since the absorption routines are also required for any Ohmic contacts in the simulation.

A relatively simple model is used for Ohmic contacts. At an Ohmic contact, the potential is defined exclusively by the externally applied potential. Unlike a Schottky barrier contact, there is no offset voltage. An Ohmic contact is capable of both injecting and absorbing carriers (in this case electrons). The injection is accomplished by adding particles at the end of each simulation time step to force the region immediately under the contact to be charge neutral. Charge neutrality is only forced when more majority carriers are needed under the contact. It is assumed that if there is already a surplus of majority carriers, then some will quickly be removed by reaching the boundary and being absorbed out of the simulation.

A brief discussion of the carriers is helpful at this point. The number of carriers in the simulation is not a fixed quantity. In fact, in the case of the MESFET simulation, the total number of carriers in the device decreases by about 20% during the course of the simulation due to the formation of the depletion region under the gate. The charge per carrier is considered to be fixed. It is defined at the beginning of the simulation to be

$$Q_{PARTICLE} = \frac{Q_{FIXED}}{N}, \tag{2.32}$$

where Q_{FIXED} is the total fixed charge in the device and N is the total number of carriers initially in the simulation.

When charge neutrality is enforced under the ohmic contacts, the number of carriers injected will most closely neutralize the fixed charge in that region. Hence, when charge neutrality is enforced, the resulting net charge under the contact will be in the range $[\frac{-Q_{PARTICLE}}{2}, \frac{Q_{PARTICLE}}{2}]$. Note that for the condition of charge neutrality to be successfully maintained, the simulation time step must be small. The carriers should only be able to move a fraction of the thickness of the charge neutral region in a given time step.

Finally, there is another property of both the Schottky and Ohmic ontacts that is important. Since the electric field is discontinuous at the metal semiconductor interface, there must be an interface charge there. When computing the terminal currents, it is essential that changes in this charge be accounted for. If this is not done, then Kirchoff's current law will be violated for the device. The stored charge is computed as

$$Q_{Interface} = \epsilon \int\int_{Contact} E_z(x, y, t)dxdy. \tag{2.33}$$

This effect was found to be particularly important with the gate contact of the simulated MESFET.

2.2.5 Initial Distribution

To begin the simulation, an initial distribution must be assumed for the particles in the device. For the simulation presented here, the state of a particle is determined by four variables: position, wavevector, conduction band valley and flight-time remaining. Initially these variables are chosen in a manner consistent with the device being under no applied biases with charge neutrality enforced everywhere in the device. This distribution is neither unique, nor the best distribution in terms of time required to for the device to reach steady state. It is, however, quite simple to implement and works acceptably well. A better alternative is to use the final state from a previous simulation as the inital state for future simulations. Typical initial and final spatial distributions are shown later as figures 4.5 and 4.3. Initially all the particles are assigned to be in the Γ valley since with no applied bias, over 99% of the carriers reside there.

The initial positioning of the particles for the condition of no applied bias seems very simple. The probability of a particle being at a given position in the device should depend only on the doping concentration in that region. Hence, a random distribution weighted according to the doping concentration should provide a reasonable initial distribution. This method was literally implemented and was found to cause a problem. Since each particle carries a charge much larger than that of an electron, any accumulation or depletion of particles in the initial distribution of particles results in artificially high local electric fields. The problem manifested itself in the form of electrons with energies greater than 4eV which is unrealistic for the simulated device and bias conditions.

The solution to the problem required two changes. First, the charge assignment technique was changed from the "nearest grid point" method to the "cloud-in-cell" method. See section 2.3.1 for more details regarding these methods. The second part of the solution required smoothing out the distribution of particles. This smoothing was accomplished by assigning the number of particles that would most closely neutralize the fixed charge associated with a point in the grid which is used for solving Poisson's equation. See section 2.3.1 for more details regarding the mesh. These two items, in combination, removed the problem of locally high electric fields.

When the initial wavevector for a particle is selected, only the magnitude is constrained; the direction is chosen at random. The magnitude is selected according to the Maxwell-Boltzmann distribution for the given lattice temperature. These assumptions are again consistent with the assumption of no applied bias. The procedure for selecting $|\vec{k}|$ is more easily applied to the energy of the particle rather than the wavevector directly. Once the energy is determined, the wavevector can easily be determined by

$$|\vec{k}| = \frac{\sqrt{2mE}}{\hbar}. \tag{2.34}$$

A brief derivation of how the Maxwell-Boltzmann distribution was implemented is in order at this point. The density of electrons residing at an energy, E, is given by

$$n(E) = f(E)g_c(E), \tag{2.35}$$

where n is the density of electrons, f is the Fermi function, and g_c is the density of conduction band states. Assuming nondegenerate statistics, the Fermi function is given by

$$f(E) = \exp\left\{\frac{E - E_F}{k_B T}\right\}, \tag{2.36}$$

where E_F is the Fermi energy, k_B is the Boltzmann constant, and T is the absolute temperature. The density of states for the conduction band is given by

$$g_c(E) = \frac{(m_n^*)^{3/2}\sqrt{2}}{\pi^2\hbar^3}(E - E_C)^{1/2}. \tag{2.37}$$

Assigning $E_C = 0$,

$$A = \frac{(m_n^*)^{3/2}\sqrt{2}}{\pi^2\hbar^3} \tag{2.38}$$

and substituting 2.36, 2.37, and 2.38 into 2.35 yields

$$n(E) = A\sqrt{E}\exp\left\{\frac{E_F - E}{k_B T}\right\}. \tag{2.39}$$

It is advantageous to normalize 2.39 with respect to its maximum value. Taking the derivative of equation 2.39 and setting it equal to zero to find the value of E for which n is a maximum yields:

$$E = \frac{k_B T}{2}. \tag{2.40}$$

Substituting back gives,

$$n_{max} = A\sqrt{\frac{k_B T}{2}} \exp\left\{\frac{E_F - \frac{k_B T}{2}}{k_B T}\right\}. \tag{2.41}$$

Normalizing gives,

$$p(E) = \frac{n(E)}{n_{max}} = \sqrt{\frac{2E}{k_B T}} \exp\left\{0.5 - \frac{E}{k_B T}\right\}. \tag{2.42}$$

This probability is plotted for $T = 300K$ in figure 2.9.

Random energy values may be selected according to the distribution shown in figure 2.9 using a rejection technique like that given in [3]. First an energy in the range $[0, E_{max})$ is chosen. A second random number in the range $[0, 1)$ serves as an acceptance factor. These numbers corresponds to selecting a point at random on figure 2.9. Once the point is selected, it must be compared with the probability distribution. If the point lies above the curve, then it is rejected. Otherwise it is accepted and the energy is considered valid for the particle. When the energy has been selected, the magnitude of the wavevector is calculated according to equation 2.34. The polar angles ϕ_1 and γ_z (see figure 2.8) are chosen so that the direction is uniformly random. Finally, k_x, k_y, and k_z are calculated by converting from spherical to Cartesian coordinates.

Although specific method used for selection of $\left|\vec{k}\right|$ is not particularly important because of the very small amount of computer time it uses, it is worth noting that the rejection is accomplished by the combined technique given in Appendix B of [3]. This technique involves using the line AB (shown in figure 2.9) to make the first cut at rejection. This first cut is done because the equation for AB can be much more quickly than the probability distribution.

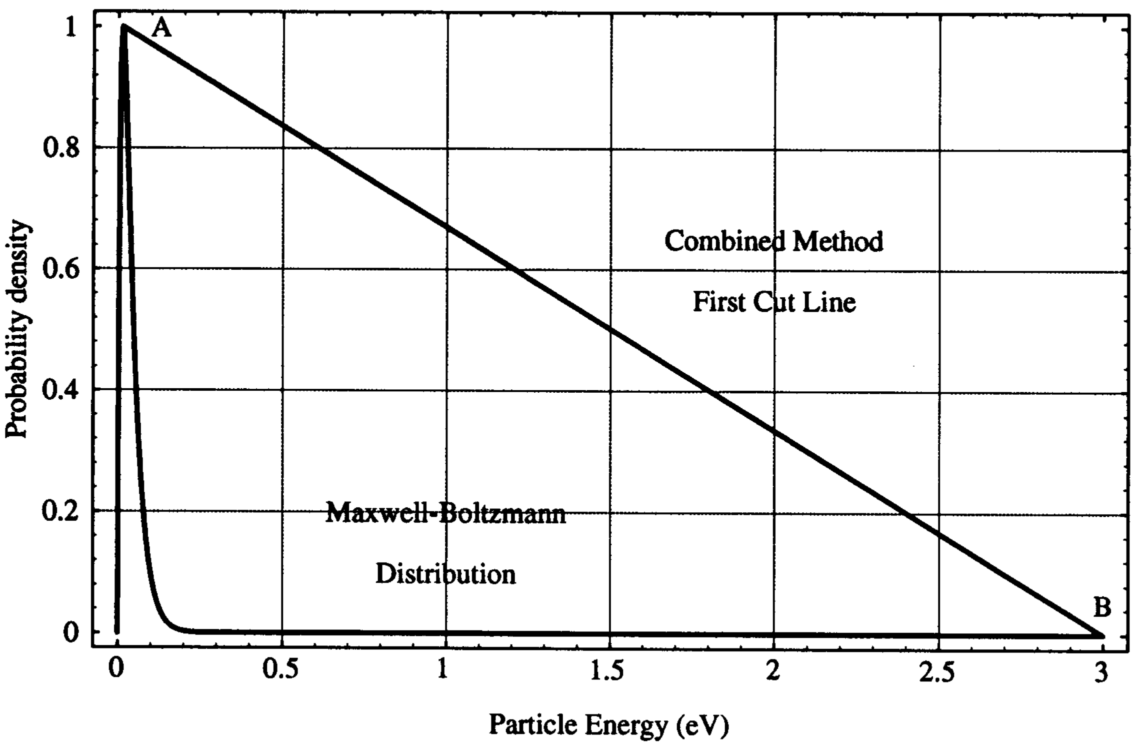


Figure 2.9: Maxwell-Boltzmann probability distribution for electrons at 300K.

2.2.6 Random Number Generation

As the name Monte Carlo implies, the generation of psuedo random numbers is an essential part of any simulation of this type. Ideally, the pseudo random number generator should meet four criteria. First, it should have no correlation to the simulation program. This criterion means that the simulation program should give statistically the same results regardless of the psuedo random number generator used. Second, it should contain no sequential correlations. Third, it should have an infinite period, i.e. it will never repeat itself. Finally, it must be very fast. With these criteria in mind, an algorithm was selected from [6].

The algorithm is based on the use of three integer linear congruential generators. A linear congruential generator is based on the formula

$$r_{i+1} = Modulo(Ar_i + C, M), \tag{2.43}$$

where A,C and M are constants. The use of a single linear congruential generator is inappropriate in the case of Monte Carlo. It will contain sequential correlations in the low order bits, and have a short period compared to the 10^8 random numbers typically required. To correct these problems, the chosen algorithm uses one linear congruential generator to generate the high order bits, and one to generate the low order bits. The use of two linear congruential generators in this manner has the effect of extending the period and reducing the sequential correlation. A third linear congruential generator is used to control a shuffling array. By shuffling the random numbers, the statistical correlation is further reduced, and the period becomes effectively infinite. It may seem like the use of three linear congruential generators would be prohibitively slow. This is not the case, since most of the calculations are performed on integers.

In addition to the four conditions above, the random number generator chosen must also be parallelizable so that it will not be a bottleneck when the code is run in parallel. This topic is discussed in section 3.3.1.

2.3 Potential Solution

As described in section 2.2.3, an electric field is present which accelerates the carriers. The electric field is obtained from the electrostatic potential distribution determined by numerically solving Poisson's equation. There are four steps in the process of determining the electric field. First, the solution region is discretized into a three-dimensional grid of points. Second, the point charges of the particles are assigned to the grid points in order to calculate the charge density distribution (section 2.3.1). Third, Poisson's equation is solved numerically on the same grid used for charge assignment (section 2.3.2). Finally, the electric field is calculated at arbitrary points as it is needed from the gradient of the potential.

The grid used for discretization has two requirements placed on it in the simulation presented here. First, the grid is defined by the set of points common to three mutually perpendicular sets of parallel planes. The parallel planes within a set are not required to be uniformly spaced. There are, however, some considerations regarding how much the plane-to-plane spacing should vary in one direction

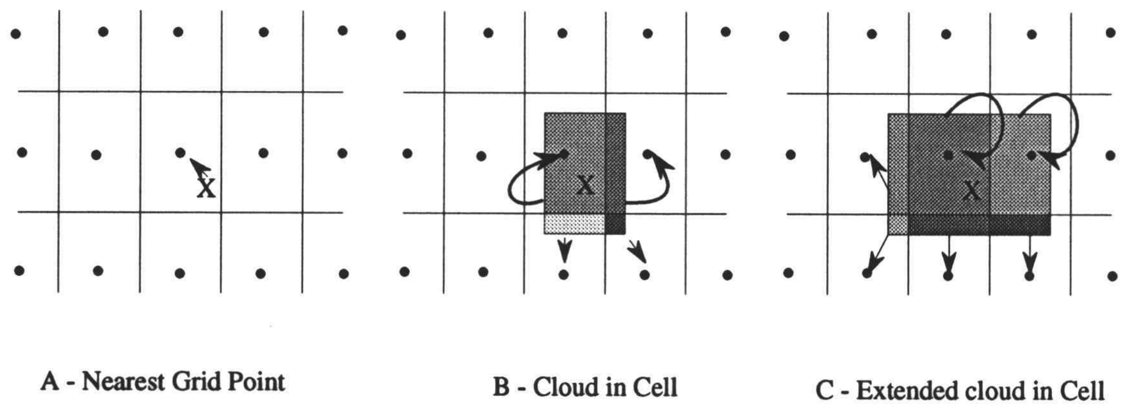


Figure 2.10: Charge assignment methods.

discussed in [2]. Second, boundaries of the simulation region must include the outermost points of the grid.

2.3.1 Charge Assignment

To utilize the difference equation to be derived in section 2.3.2, the charge density must be determined at each point in the mesh. There exist two basic methods for accomplishing this task. In both methods, the fixed charge is assigned to the grid point to which it is nearest. The simplest method is known as the "nearest grid point" method. The other is known as the "cloud-in-cell" method. Recently, there has been an interesting variation on the cloud-in-cell method used by [7]. The three methods are diagramed in figure 2.10.

With the nearest grid point method, the charge carried by each particle is assigned to the grid point closest to the particle. The sum of the fixed and mobile charges assigned to a point is then divided by the volume defined by all points where it is the nearest grid point. Since the mesh points are set up in a regular array, this volume is defined by a box whose sides are half way to the six planes of points around the given point. This method was used initially and was found to create two important problems. First, it was possible to obtain too much charge assigned to a single grid point, resulting in unphysically high local electric fields for the simulated device and bias conditions. The second problem is that a small change in the position of a particle can cause a significant change in the local charge density distribution. This change occurs when the particle crosses a boundary between two mesh points.

The cloud-in-cell method uses a box around the particle to smear out the charge over a region. The fraction of the box that overlaps with the volume assigned to a grid point is the fraction of the charge that will be assigned to that grid point (see part B of figure 2.10). In the case of non-uniform grid spacing the eight surrounding grid points are used. For non-uniform grid spacing, this distribution creates problem. The effective charge density represented by a particle will change with grid spacing. An "extended cloud-in-cell" method, which cures this problem, has been used by [7].

The extended cloud-in-cell method uses a fixed cloud (box) size about the particle. The charge associated with the particle is then assigned to each grid point whose region overlaps with the cloud in proportion to the amount of overlap. This method is shown in part C of figure 2.10. The extended cloud-in-cell method has none of the problems associated with the nearest grid point method or the cloud-in-cell method.

2.3.2 Solution of Poisson's Equation

Solving Poisson's equation is an integral part of any semiconductor device simulation. Poisson's equation is written in mks units as:

$$\nabla^2 V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = \frac{-\rho(x,y,z)}{\epsilon}, \tag{2.44}$$

where V is potential, $\rho(x,y,z)$ is charge density, and ϵ is permittivity. The solution of such a differential equation is referred to as a boundary value problem. There are two types of boundary conditions possible at every grid point on the edge of the device. Either the potential, V, is explicitly defined, or the electric field is defined (usually to be zero). The usage of these conditions is discussed in section 2.2.4.

Since, in general, it is not possible for a computer to obtain closed form solutions to equation 2.44, approximate methods must be resorted to. The first

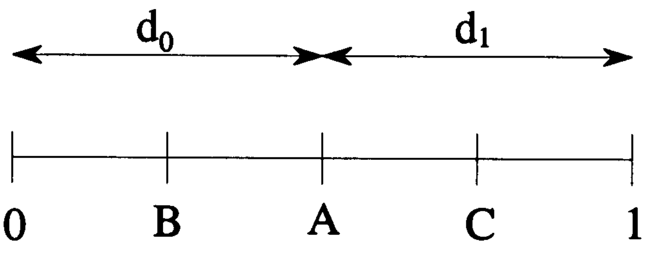


Figure 2.11: Definition of terms for finite difference method.

approximation is to only determine the solution for V at the discrete grid points. It will prove useful if the grid is split into two categories. Those points where the sum of the array indices is even are referred to as red points and those points where the sum is odd are referred to as black points. This process produces a three dimensional checkerboard pattern. Any red point is surrounded by black points and any black point is surrounded by red points.

Now that the set of solution points is defined, an approximation to Poisson's equation must be found. For the program developed here, the finite difference scheme was chosen. The derivation presented here introduces the notation used in sections 3.3.2 and 3.4. Part of the notation used is defined in figure 2.11. In figure 2.11, point B is defined to be the midpoint of $\overline{0A}$ and point C is defined to be the midpoint of $\overline{A1}$.

Applying the difference equation to the x component of Poisson's equation gives

$$\frac{\partial^2 V}{\partial x^2} = \frac{\left.\frac{\partial V}{\partial x}\right|_C - \left.\frac{\partial V}{\partial x}\right|_B}{\frac{1}{2}(d_0 + d_1)}. \tag{2.45}$$

Applying the approximation a second time gives

$$\frac{\partial^2 V}{\partial x^2} = \frac{\frac{V_1 - V}{d_1} - \frac{V - V_0}{d_0}}{\frac{1}{2}(d_0 + d_1)} = 2\left[\frac{V_0 - V}{d_0(d_0 + d_1)} + \frac{V_1 - V}{d_1(d_0 + d_1)}\right]. \tag{2.46}$$

The subscripts 2 and 3 are similarly used for the y direction and 4 and 5 for the z direction. Poisson's equation becomes, in the difference approximation,

$$\nabla^2 V = \frac{-\rho}{\epsilon} \cong 2\left[\frac{V_0 - V}{d_0(d_0 + d_1)} + \frac{V_1 - V}{d_1(d_0 + d_1)} + \frac{V_2 - V}{d_2(d_2 + d_3)} + \frac{V_3 - V}{d_3(d_2 + d_3)} + \frac{V_4 - V}{d_4(d_4 + d_5)} + \frac{V_5 - V}{d_5(d_4 + d_5)}\right]. \tag{2.47}$$

To simplify the notation, the following substitutions are made

$$\begin{aligned} t_0 &= [d_0(d_0+d_1)]^{-1} \quad t_1 = [d_1(d_0+d_1)]^{-1} \\ t_2 &= [d_2(d_2+d_3)]^{-1} \quad t_3 = [d_3(d_2+d_3)]^{-1} \qquad (2.48) \\ t_4 &= [d_4(d_4+d_5)]^{-1} \quad t_5 = [d_5(d_4+d_5)]^{-1} \\ t_{012345} &= \sum_{i=0}^{5} t_i. \qquad (2.49) \end{aligned}$$

Substituting into equation 2.47 and rearranging gives

$$t_{012345}V = \frac{\rho}{2\epsilon} + t_0V_0 + t_1V_1 + t_2V_2 + t_3V_3 + t_4V_4 + t_5V_5. \qquad (2.50)$$

Using equation 2.50, a system of simultaneous equations can be written. The most obvious idea is to write the coefficient matrix and invert it. The coefficient matrix is very sparse and will not occupy a significant amount of memory. Unfortunately, its inverse is not sparse and would require a huge amount of memory. If there are 8000 points to be solved for, the inverse matrix would require in excess of 512 megabytes of memory. Although this is not currently practical, it may be a viable alternative one day.

Since a direct solution technique is not possible, an iterative technique known as Simultaneous Over-Relaxation (SOR) was chosen. This technique has minimal memory requirements (64 kilobytes for 8000 points) and is acceptably fast. In addition, it is both parallelizable and vectorizable. These topics will be discussed in sections 3.3.2 and 3.4. The SOR method is an iterative process that consists of alternately solving for the potentials at the red points with the potentials at the black points held constant and vice versa. The iteration formula used at each step is a weighted average of the potential given by solving equation 2.50 and the potential from the previous iteration over the current color. Mathematically, this iteration formula is written as

$$V_{New} = (1-\omega)V_{Old} + \omega V, \qquad (2.51)$$

where ω is known as the relaxation factor.

If a relaxation factor less than unity is chosen, the resulting effect is termed under-relaxation. Similarly, a relaxation factor greater than unity is said to produce over-relaxation. It has been shown ([6]) that over-relaxation is always faster than under relaxation. The over-relaxation factor is not constant and is updated according to the prescription

$$\begin{aligned} \omega^{(0)} &= 1 \\ \omega^{(1/2)} &= \frac{1}{1-\frac{\rho^2_{Jacobi}}{2}} \\ \omega^{(n+1/2)} &= \frac{1}{1-\frac{\rho^2_{Jacobi}}{4}} \quad n = 1/2, 1, \cdots, \infty \\ \omega^{(\infty)} &\longrightarrow \omega_{optimal}. \end{aligned} \tag{2.52}$$

This scheme is known as Chebyshev acceleration (see reference [6] for more details). Through experimentation, a good value for ρ_{Jacobi} was found to be 0.99970.

The SOR method is implemented by calculating residuals. The residual at a point is defined to be

$$\varepsilon = \frac{\rho}{2\epsilon} + t_0V_0 + t_1V_1 + t_2V_2 + t_3V_3 + t_4V_4 + t_5V_5 - t_{012345}V_{Old}. \tag{2.53}$$

V_{New} in terms of the residual is given by

$$V_{New} = V_{Old} + \omega\frac{\varepsilon}{t_{012345}}. \tag{2.54}$$

This particular method of computation was chosen because the residuals are also a convenient way to determine if convergence has been reached.

The test used to detect convergence is based on the idea of globally reducing the residuals. After each full iteration, the sum of the absolute value of the residuals is calculated. The condition used to detect convergence is a total reduction by a factor 0.001 in the residual. This idea is written as

$$\frac{\sum_{all}\varepsilon}{\sum_{all}\varepsilon_{initial}} < 0.001. \tag{2.55}$$

This test has been found to be a reliable criterion for convergence. The complete process of solving Poisson's equation is presented in figure 2.12.

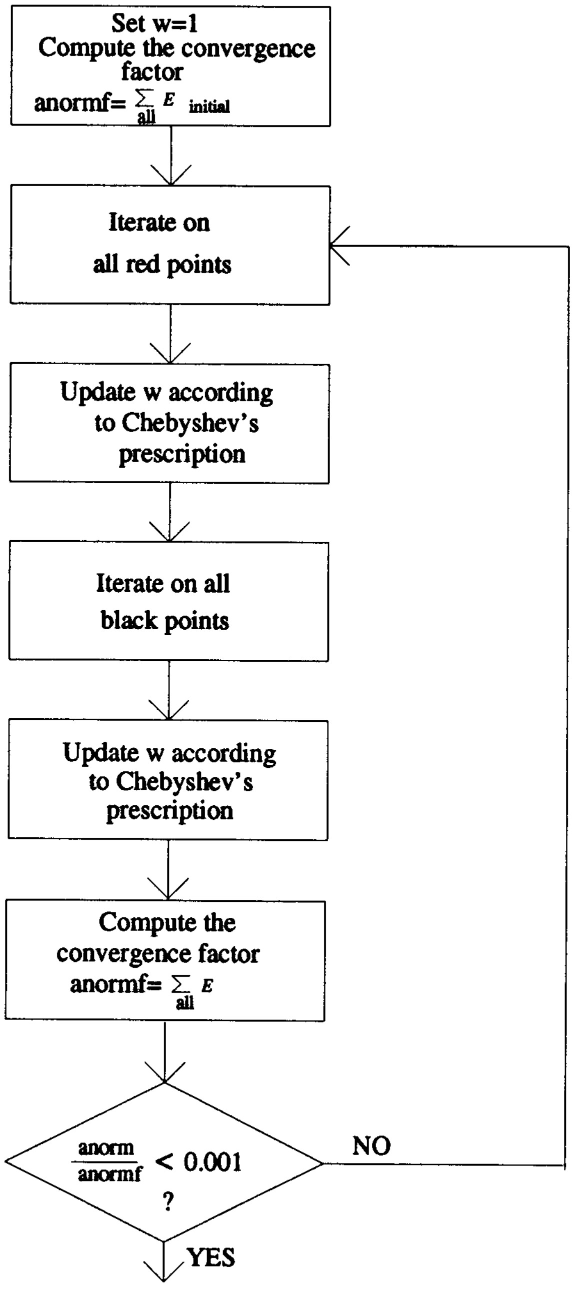

Set w=1
Compute the convergence factor
anormf= $\sum_{all} E$ initial
Iterate on all red points
Update w according to Chebyshev's prescription
Iterate on all black points
Update w according to Chebyshev's prescription
Compute the convergence factor
anormf= $\sum_{all} E$
$\frac{anorm}{anormf} < 0.001$?
NO
YES

Figure 2.12: Flowchart of solution to Poisson's equation.

Chapter 3

Formulation of the Problem

When designing any simulation program, one must consider the capabilities of the computer the program is intended to run on. The speed and memory capabilities of the machine are primary considerations when deciding what approximations and assumptions are to be used. This fact is especially true for Monte Carlo solid state device simulations. Monte Carlo simulations have traditionally had several limitations imposed by computer capabilities. These limitations included simplified band structures (parabolic bands), two dimensionality, single carrier type and single material type (no quantum effects). Even with these constraints, Monte Carlo simulations are still extremely computer intensive. In some recent publications ([8]) authors have given execution times on mini-supercomputers that are measured in hours. It is obvious from such times that a large increase in computational requirements cannot be accommodated by such computers.

One approach to accommodate future computational requirements is to purchase machines that are sequentially faster. Unfortunately, the cost of such machines is exponentially related to the speed. The alternate approach being considered here is the use of parallel computers. There are two primary advantages of parallel computers over fast sequential computers. First, the cost of parallel computers tends to be linear with the computational power of the machine. Hence, such machines will be less expensive than their sequential counterparts. Second, the parallel computer may be able to achieve computational speeds not presently available on sequential machines at any price. For these advantages to be of any benefit, the problem of interest must be suitable for parallelization. The advantages of parallel computers are offset by one major disadvantage. The software development tools currently available on parallel computers are not very advanced. Efficient parallel programming still depends largely on the efforts and creativity of the programmer.

3.1 Parallel Computers Applied to Monte Carlo Simulation

It was noted above that the benefits of using a parallel computer are only realized if the problem being considered is parallelizable. For the Monte Carlo device simulation discussed in chapter 2, both the particle movements and the solution of Poisson's equation are individually parallelizable.

In the case of the particle flights, it is easy to see the parallelism that is present. The particles interact with each other only through the electric field. First, the electric field is determined using the charge distribution. Then the electric field that was determined is used to accelerate the particles for a short time step. The important consequence of this is that during a simulation time step, all of the particle flights are independent. Since they are independent events, they can be processed simultaneously by different processors.

With the derivation in section 2.3.2, the parallelism in the solution of Poisson's equation is also fairly easy to see. It was noted in section 2.3.2 that the red-black ordering scheme was chosen. More importantly, it was also noted that during an iteration over the red or black points, the other color is held constant. Thus, the solution at a given point does not directly depend on any other points of the same color. Hence, the points considered in any half sweep can be processed simultaneously by different processors. It should be noted from this brief discussion that it is essential for all the processors synchronize between half sweeps. Otherwise one processor might still be working on red while the others have gone onto black. Synchronization is discussed in section 3.3

There are additional parts to the solid state device simulation such as the generation of the scattering rate tables. Some of these parts of the simulation may be parallelizable. However, due to the tremendous effort required and the very small speed increase possible in this case, no parallelism was implemented in these portions of the program. If the number of processors were increased substantially, then these sequential routines would eventually become a bottleneck. At such a point, it would be worth considering them for parallelization.

3.2 Ardent Titan 1500 System

Parallel computers usually fall into two categories based on how memory is accessed. The memory can be shared such that all of the processors can directly access it. Machines of this type are referred to as shared memory parallel computers. The other type of architecture has memory exclusively associated with each processor. These machines are referred to as distributed memory parallel computers. The computer used for the simulation presented here was an Ardent (Stardent/Kubota Pacific Computer) Titan II. The Titan II is a shared memory parallel computer with four processors. An alternate study of the implementation of this simulation on a distributed memory parallel computer was performed by Udaya Ranawake at Oregon State University [9].

The Titan II has the additional capability of efficiently processing vectors. Each of the four processors is equipped with a vector processing unit. These units are capable of most simple comparison and arithmetic functions, except division. There are three important facts about the shared memory and vector units that must be presented here. First, if stride-one (see [10, 11]) access to memory is achieved, then the maximum access rate to the shared memory is 64MHz. The Ardent's memory is arranged in 16 interleaved banks. Stride one access refers to accessing these banks in a rotating manner, 0, 1, 2, ... 15, 0 ... Stride-one access is the most efficient way to access memory in the Ardent. Second, each vector processor (see [10]) is capable of accessing memory at 32MHz. Third, the processors themselves can only access memory at 16MHz. From these numbers, it is obvious that there will be a problem with heavy memory access. For example, suppose all four vector processors are attempting to access memory at 32MHz. This situation implies a total access rate of 128MHz is being attempted when only 64MHz is possible. A second example is also worth presenting. Suppose the processors are all trying to access memory in some random manner. In this situation, stride-one access is not maintained, significantly decreasing the possible memory access rate. Hence, the memory is again a bottleneck. These two examples are typical of the Poisson solution and the particle flights, respectively.

The next important item regarding the Titan II is the interprocessor communication facilities that are available. Interprocessor communication may seem like a simple matter since all the memory is accessible to all the processors. It is, in fact a very difficult problem. The problem is that the processors must communicate in order to synchronize when necessary. In the case of the Titan II, this synchronization is accomplished with the use of semaphores. Semaphores are boolean variables that should only be set or cleared by an atomic operation known as locking. Because of the atomic nature of the modification, erroneous read-modify-write operations such as the one shown in figure 3.1 are not possible. In figure 3.1, the modification to x by processor 2 is lost. Figure 3.2 shows how this can be avoided with the use of a semaphore. All synchronization problems in the present work were solved with the use of semaphores. Synchronization will be discussed in more detail in sections 3.3.1 and 3.3.2.

Finally, a word about the compiler and development tools is in order. The Titan II compilers are able to recognize some opportunities for parallelization in simple cases. Unfortunately, it is far beyond the capabilities of these compilers to recognize high-level parallelism in C. In fact, they will not try to parallelize anything that uses pointers. This fact means that essentially all the parallel considerations are the programmer's responsibility. The compiler only provides certain tools like shared variables and the atomic locking operation mentioned above.

The compiler is not much better with recognizing vectorization opportunities.

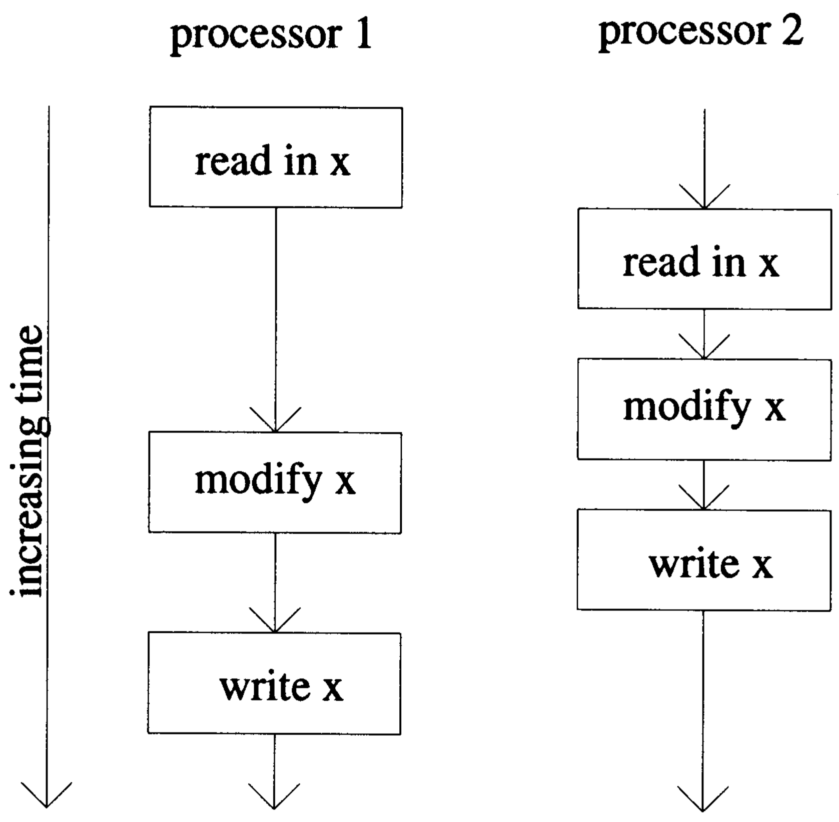


Figure 3.1: Unsynchronized access to a shared variable.

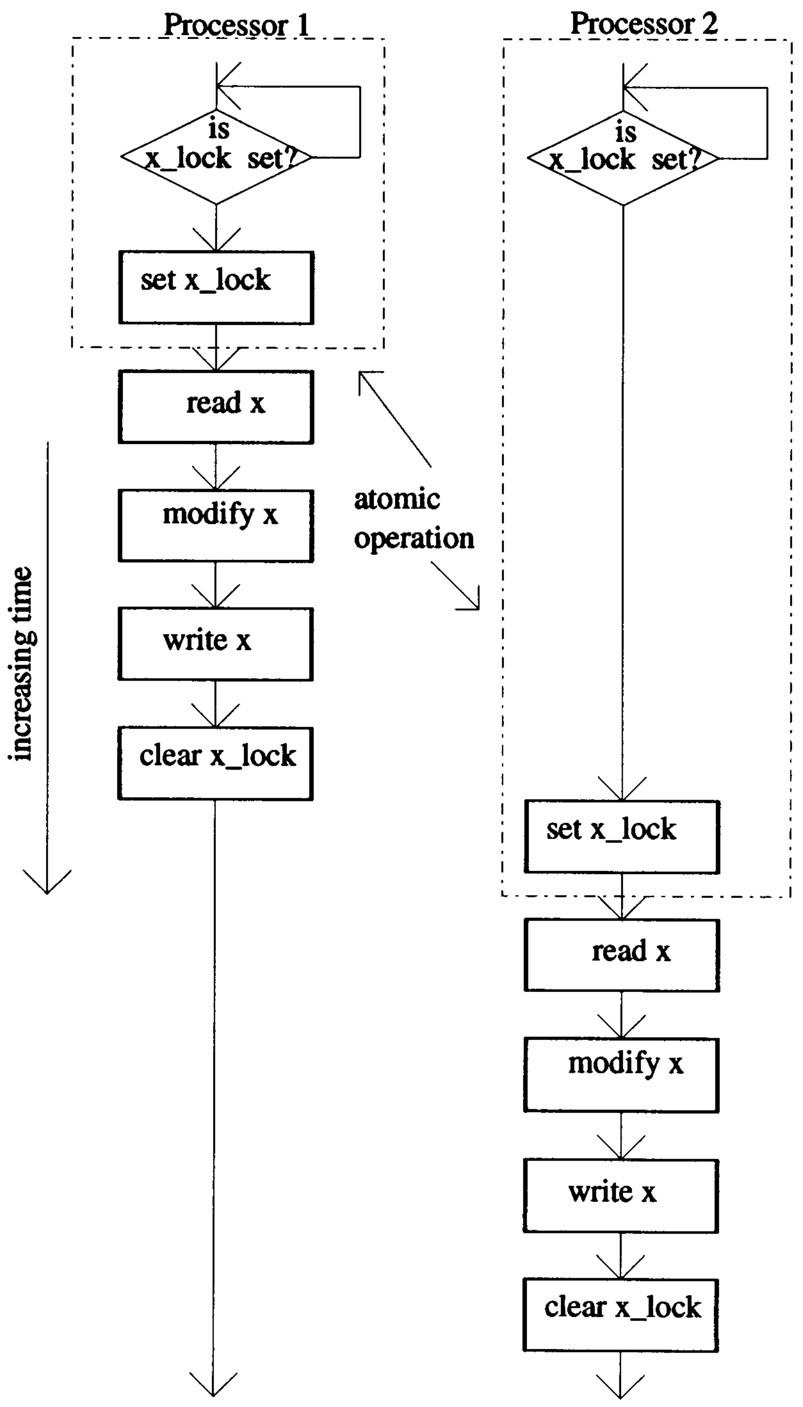


Figure 3.2: Synchronized access to a shared variable.

Again, if there are pointers involved, the compiler must be forced using a directive to implement vectorization. Although the Titan Programmer's Guide [10] discusses how to use compiler directives, it is a very poor manual. Most of the information needed to accomplish both vectorization and parallelization came from the support staff at Stardent Computer Inc. [11].

3.3 Parallelization

There are three general topics which should be discussed before proceeding to a detailed discussion of how parallel aspects of the Monte Carlo simulation were implemented. First, the concept of the level of parallelism used should be discussed. Usually three levels of parallelism are defined. The lowest level is referred to as fine grain parallelism. This form is applicable to virtually all programs on shared memory computers, but is not as effective as the next two levels. Usually fine grain parallelism is implemented by the compiler without assistance from the programmer. The Titan II compilers implement fine-grain parallelism in this manner.

The second level of parallelism is referred to as medium grain parallelism. This form is usually thought of as applying to loops where each iteration is clearly independent of other iterations. Depending on the complexity of the loop contents, and the sophistication of the compiler, this parallelizing may be done automatically. Often direct intervention from the programmer is required to recognize the opportunity for parallelization. Intervention is usually required with the Titan II compilers.

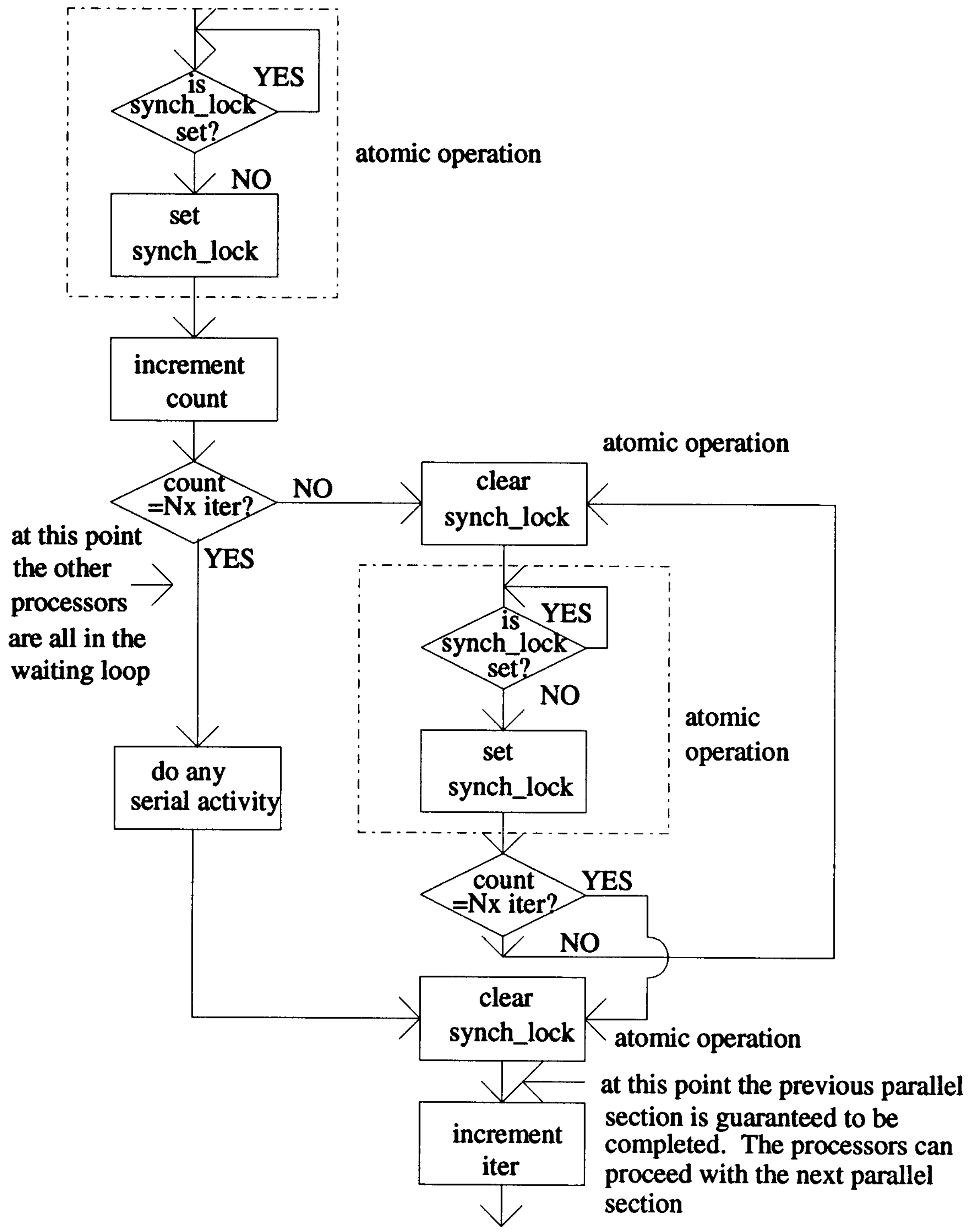
The last level is referred to as course grain parallelism. Typically, large blocks of the program are independent. This level is never achieved by a compiler without significant help from the programmer. However, as one might suspect, it is the most efficient form of parallelism.

The second topic is local and shared variables. Synchronization of accessing a shared variable is discussed in section 3.2. There is another method for protecting some shared variables which does not rely on semaphores. If each processor can be assigned a task before the program begins parallel execution, then competition for some shared variables may be avoided entirely. However, this case is the exception, not the rule. In the case of variables used in subroutines, either semaphore protected shared variables or local variables are necessary. Local variables are those that are allocated in memory reserved for use by only one processor. In C these variables can take several forms. They can be dynamically allocated as the function is entered, or by a semaphore protected call to malloc, the UNIX system memory allocator, or by a threadlocal variable declaration. The full set of declarations possible is too large to describe here. The interested reader can refer to [10] for more details.

The final topic before addressing the details of implementation is synchronization. At some points in the program, synchronization is necessary to synchronize all the processors. Since the only efficient means of interprocessor communication are semaphores and shared variables, it is a non-trivial problem. The solution that was devised is presented in figure 3.3. The solution depends on the use of a shared variable that counts the total number of processor-iterations completed. A processor-iteration is defined to be the work required of one processor before synchronization is required. The definition of a processor-iteration will be discussed further in section 3.3.2. Each processor also tracks how many processor-iterations it has completed. Synchronization is accomplished by only letting the processors proceed when the toal number of processor-iterations is greater than or equal to the number of processor-iterations completed by the processor multiplied by the number of processors. Put simply, it is a way to verify that all the other processors have completed their share of the work.

3.3.1 Acceleration and Scattering

As previously noted in section 3.1, within a scattering time step, all particle flights are independent. Each particle may be arbitrarily assigned to any of the processors for processing. Hence, other considerations will determine which processor a particle is assigned to. The first, and most important, of these considerations



N = number of processors (constant)

iter = iteration number (local variable)

count = the total number of processor iterations completed (shared variable)

synch_lock = a semephore used for protecting count

Figure 3.3: Synchronization of all processors.

is load balancing. The second consideration is the data structure used to hold the particle information.

Load balancing is extremely important on the Titan II. A brief anecdote will serve to illustrate this point. One of the test versions of the Monte Carlo simulation attempted to balance the load by allocating one fourth of the particles to each processor. This scheme worked acceptably well if there were no other jobs being run on the system. When there was an additional one processor job running on the system, then the four processor parallel simulation took longer to run than the serial simulation. Obviously there must be some more intelligent form of load balancing incorporated in the simulation directly.

The technique implemented in the present work divides the particles up into groups. The processors continually choose groups for processing until there are no more groups to be processed. With this algorithm, the idle time of any of the processors is usually kept quite small. There is one unusual and interesting circumstance that has been observed to cause a slight delay in program execution. If three of the four processors have acknowledged completion of all groups, and the fourth one is extremely busy (e.g. running the tape drive), then there may be a pause before the fourth processor acknowledges completion. Again, this event is quite rare and is not very significant.

The choice of groups comes from the data structure that contains the particles. The particles are contained in linked lists that originate in the structure of the grid point they are closest to. Through experimentation, a reasonable group was found to be all particles attached to grid points in a line in the x direction. This division provides a good trade off between the overhead of dealing with a large number of groups and having enough groups to make the load balancing effective. The groups are implemented with one shared boolean variable per row to indicate whether it requires processing. The ability to search these flags is protected by a semaphore.

It is useful while discussing parallelizing the acceleration and scattering to discuss one scattering mechanism in detail as an example. This discussion may prove helpful to anyone attempting to add additional scattering mechanisms in the future. The sample function is presented in figure 3.3. The function declaration is a standard C declaration. The variable declarations must all be preceded by a threadlocal declaration. A threadlocal declaration guarantees that each processor has a private variable to work with. Without such a declaration, unpredictable results will occur. The #pragma is a compiler directive that tags the function as safe for parallel execution. See [10, 11] for more information about this. Finally, note that the function finalk is called by the shown function. Any function called by a parallel function must also be tagged as safe for parallel execution.

As a closing note about acceleration and scattering, the generation of random numbers should be briefly addressed. Two approaches were looked at. First, the random number generator described in section 2.2.6 was protected by a semaphore. This technique of having four processors share one random number generator required about 10% of the total processor time to generate random numbers. The second approach was to give each processor a copy of the random number generator with different seeds. This approach was found to reduce the processor time required for random number generation to about 1.5% of the total processor time. The reduction is better than a factor of four because there is no overhead for semaphores. The percentages given here are approximations based on output from the profile option of the compiler.

3.3.2 Solution to Poisson's Equation

Parallelization of the solution to Poisson's equation was accomplished in much the same manner as for acceleration and scattering (see section 3.3.1). The load balancing considerations are essentially the same in either case. Since each point iterated on during a half sweep is independent, they can be arbitrarly grouped. As with acceleration and scattering, the solution groups are lines of points in the x direction. The processors continue to choose groups for processing until all points

```
/*
 Acoustic:
 Acoustic scattering is elastic and isotropic and the azimuthal angle,
 theta is calculated using the direct technique.
*/
void acoustic(particle,dummy)
PARTICLE *particle;
INT32 dummy;
{
 threadlocal FLOAT64 eki,ki,kf,phi,theta;
#pragma pproc acoustic
/* Compute energy of particle */
 eki=PARTICLE_ENERGY(particle);
/* Compute magnitude of wave vectors */
 ki=WAVE_VECTOR_MAGNITUDE(eki,particle→valley);
 kf=ki;
/* Choice of theta by the direct technique */
 theta=ACOS(1.0−2.0*ILRAND());
/* Choice of phi randomly between 0 and 360 degrees */
 phi=2.0*pi*ILRAND();
/* Get x, y and z components of final wave vector */
 finalk(particle,ki,kf,theta,phi);
 return;
}
```

Figure 3.4: Acoustic scattering parallelized function.

of the specified color have been iterated on. When there are no more points to iterate on, the processors will synchronize. While three of the processors are in the waiting loop (see figure 3.3), the fourth tests for convergence and informs the other processors through a shared boolean variable. The processor-iteration count is then incremented and all the processors are free to proceed. This sequence of events is shown in figure 3.5. The choice of groups used in parallelizing this portion of the program has the additional advantage of being highly compatible with vectorization, as discussed further in section 3.4.

3.4 Vectorization of Solution to Poisson's Equation

On the Titan II, a vector is defined as a regularly spaced sequence of double precision values. It is not necessary that these values be adjacent in memory, only regularly spaced. The vector processing units in the Titan II can efficiently perform a number of simple arithmetic operations on such vectors. Note that division cannot be performed, but operations between vectors and scalars are allowed. These vector operations are very powerful tools when an iteration according to equation 2.51 is being applied to a line of grid points parallel to any of the three coordinate axes. For a line of points parallel to an axes, all the values required occur as scalars or a regular array of double precision values. Since the groups chosen for parallelization (see section 3.3.2) are lines of grid points parallel to the x axis, these will also be the units considered for vectorization. When equation 2.51 is partially expanded, it can be written as

$$V_{New} = V_{Old} + \frac{\omega}{t_{0123456}} \left\{ \frac{\rho}{2\epsilon} + t_0 V_0 + t_1 V_1 + t_2 V_2 + t_3 V_3 + t_4 V_4 + t_5 V_5 - t_{012345} V_{Old} \right\}. \tag{3.56}$$

Now it must be determined whether each variable is a vector or a scalar in the x direction. The results are shown in Table 3.1. In the remainder of this section, vector quantities will be shown with vector notation. The astute reader may notice that there are problems with two of the vectors being short by one value. Special processing is required for points at either end in the x direction. All the

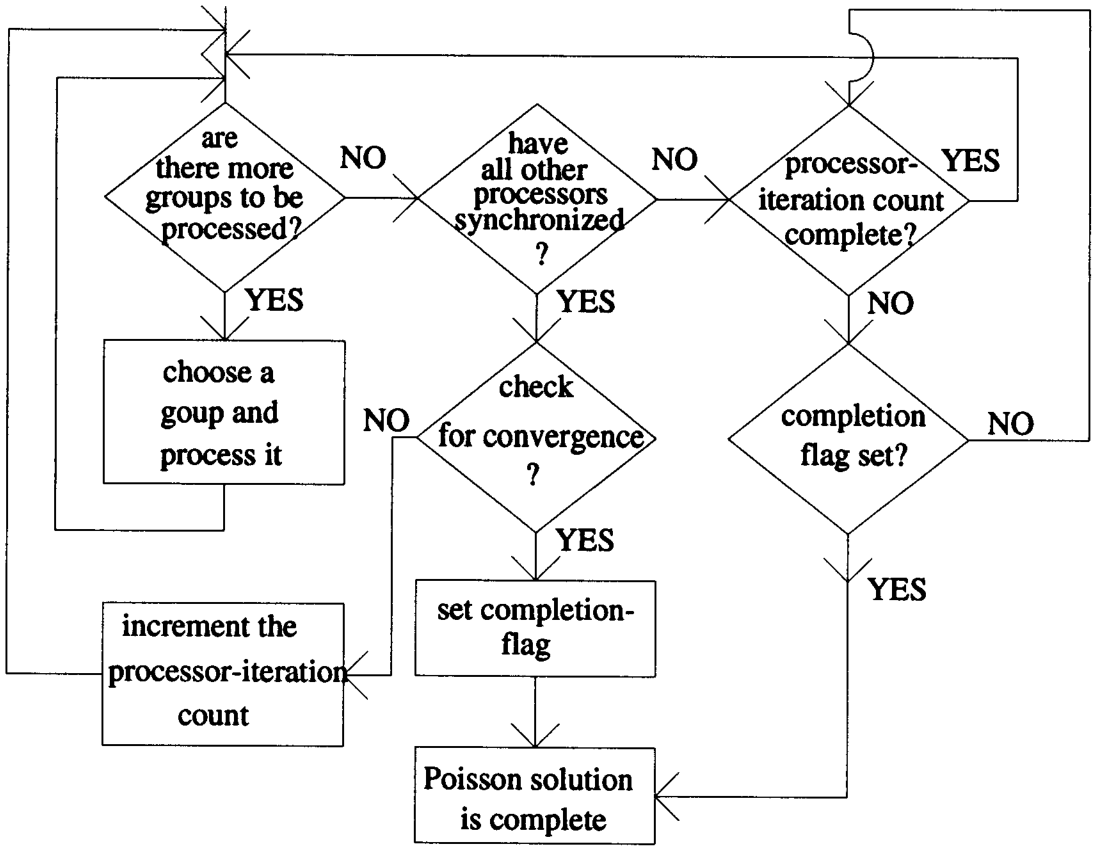

are there more groups to be processed?
NO
YES
choose a goup and process it
have all other processors synchronized ?
NO
YES
processor-iteration count complete?
YES
NO
check for convergence ?
NO
YES
completion flag set?
NO
YES
increment the processor-iteration count
set completion-flag
Poisson solution is complete

Figure 3.5: Flow chart of solution to Poisson's equation.

Neumann and Dirchlet boundary conditions on the other four sides of the device are efficiently handled in the setup for the vector loops. Beyond this attention in the loop setup, no special action is required to deal with them. The sequence of calculations performed to implement vectorization is shown in figure 3.7.

The implementation of the vector equations is extremely messy. It was necessary to use pointers, pointer arithmetic, and compiler directives to force vectorization to occur. It will undoubtedly be useful to anyone attempting to understand this code, if one of the vector loops is examined in detail. The loop presented in figure 3.6 performs the calculation $\vec{t}_{012345} = \vec{t}_0 + \vec{t}_1 + \vec{t}_{2345}$. *temp_vector* is an array of five vectors of length N. The definitions of the five vectors are given in table 3.2. *start1* represents the lowest index in the x direction to be processed for the given line. *tp1*, *tp2*, and *tp3* are temporary pointers used to step through the vectors.

Variable	Vector/ Scalar	Len	Variable	Vector/ Scalar	Len	Variable	Vector/ Scalar	Len
V_{New}	Vector	N	t_4	Scalar	-	d_0	Vector	N-1
V_{Old}	Vector	N	t_4	Scalar	-	d_1	Vector	N-1
ω	Scalar	-	t_5	Scalar	-	d_2	Scalar	-
t_{012345}	Vector	N	V_0	Vector	N-1	d_3	Scalar	-
ρ	Vector	N	V_1	Vector	N-1	d_4	Scalar	-
t_0	Vector	N	V_2	Vector	N	d_5	Scalar	-
t_1	Vector	N	V_3	Vector	N			
t_2	Scalar	-	V_4	Vector	N			
t_3	Scalar	-	V_5	Vector	N			

Table 3.1: Table of vector and scalar variables in the Poisson solver.

The compiler directive, preceding the for loop, instructs the compiler to ignore any possible dependencies and proceed with vectorization.

The actual vector loop must be of a very simple structure. It should consist of an equation (or equations) followed by each vector pointer being increased by a value that is constant within the loop. For more details regarding this loop structure, the reader should refer to [10].

Finally, the concept of stride should be revisited. Stride refers to the distance

temp_vector index	Description
1	$\vec{d}_{01}$
2	$\vec{t}_0$
3	$\vec{t}_1$
4	$\vec{t}_{012345}$
5	$\vec{\varepsilon}$

Table 3.2: Definition of temporary vectors used in the Poisson solver.

```
/*
 Calculate the vector denominator=t0+t1+t2345.
*/
      tp1=temp_vector+3*grid_dimensions[0]+start1;
      tp2=temp_vector+grid_dimensions[0]+start1;
      tp3=temp_vector+2*grid_dimensions[0]+start1;
#pragma ivdep
      for(x=start1;x<grid_dimensions[0];x++) {
        (*tp1)=(*tp2)+(*tp3)+t2345;
        tp1+=2;
        tp2+=2;
        tp3+=2;
      }
```

Figure 3.6: Example of a vectorized loop

(in terms of double words) between memory accesses. The Titan II memory boards are 16 way interleaved. That is to say, they are divided into banks with the lowest four bits of an address determining the bank. A stride value larger than 15 is equivalent to that stride modulo 16. After access to a bank, some time is required before another access to that bank can be done. This recovery time is equivalent to two access cycles. Two examples can illustrate how the stride used can affect vector unit performance. First, suppose that a vector is arranged in memory such that the stride between elements is one. This spacing will cause the vector unit to access the banks with a delay of seven access cycles between accesses to the same bank of memory. Seven access cycles represent more than enough time for the memory to recover. Now, suppose instead that the elements are spaced with a stride of 16 (or equivalently, zero). The vector processor will have to wait two access cycles between accesses to the bank. Hence, it will take three times as long to transfer the vector from the memory into the vector unit. Note, however, for the parallelized vectorized

$$\vec{d}_{01} = \vec{d}_0 + \vec{d}_1$$

$$\vec{t}_0 = \frac{\vec{d}_0}{\vec{d}_{01}}$$

$$\vec{t}_1 = \frac{\vec{d}_1}{\vec{d}_{01}}$$

Select a line of grid points if there are any remaining. If not, then synchronize.

$$d_{23} = d_2 + d_3$$

$$d_{45} = d_4 + d_5$$

$$t_2 = \frac{d_2}{d_{23}}$$

$$t_3 = \frac{d_3}{d_{23}}$$

$$t_4 = \frac{d_4}{d_{45}}$$

$$t_5 = \frac{d_5}{d_{45}}$$

$$t_{2345} = t_2 + t_3 + t_4 + t_5$$

$$\vec{t}_{012345} = \vec{t}_0 + \vec{t}_1 + t_{2345}$$

$$\vec{\varepsilon} = \left\{ \frac{\rho}{2\epsilon} + \vec{t}_0\vec{V}_0 + \vec{t}_1\vec{V}_1 + \vec{t}_2\vec{V}_2 + \vec{t}_3\vec{V}_3 + \vec{t}_4\vec{V}_4 + \vec{t}_5\vec{V}_5 - \vec{t}_{012345}\vec{V}_{Old} \right\}$$

Special processing at the extreme end points is required because $\vec{V}_0$ or $\vec{V}_1$ may not exist there. Set all elements of $\vec{\varepsilon}$ where a Dirchlet condition exists to zero.

$$norm = norm + \sum_{allelements} |\vec{\varepsilon}|$$

$$\vec{V}_{New} = \vec{V}_{Old} + \frac{\omega}{\vec{t}_{012345}}\vec{\varepsilon}$$

Synchronize and decide if more iterations are necessary.

Figure 3.7: Vector Parallel process for solving Poisson's equation.

case stride is not much of a concern. Even if a poor choice of stride should occur, the other processors will be accessing the other banks so not very much processor time is actually lost.

Chapter 4

Simulation Results

The first three sections of this chapter are included as verification of the functionality of the simulation. Section 4.1 briefly reviews the scattering rates presented in section 2.2.2 and compares them to published rates. Section 4.2 presents simulated velocity field characteristics and compares to published experimental data. Section 4.3 presents the simulated device and the current-voltage characteristics for it. These results are compared to results for a very similar device presented in [2]. Sections 4.4 and 4.5 contain speed increase results from the parallelization and vectorization of the program.

4.1 Scattering Rates

The calculated scattering rates were presented in figures 2.4, 2.5, and 2.6. Figure 2.4, the Γ valley scattering rates, can be compared to figures 2.15 and 2.17 in [3]. No published rates were found to compare with the L and X valleys. The Γ valley polar optical scattering rates compare very closely with those in [3]. In fact, there is no distinguishable difference. For the intervalley scattering mechanisms, the form of the curves are the same, but the magnitudes differ by as much as 50%. Since the formula used in each case is the same, it is likely that this difference arises from the choice of the intervalley deformation potentials. The deformation potential appears to the second power in the rate calculation and it is not a directly measured quantity. Because of these facts, it is not considered significant that the magnitudes are not identical.

No published data was found for acoustic deformation potential scattering in GaAs. The final comparison was for the total scattering rate. In figure 2.21 of [5], the total scattering rate for GaAs at 300K is presented. The form of this curve is the same as that shown in figure 2.4. A difference in magnitude of about 50% exists at an energy of 1eV. Since the same physical parameters were used, this is attributed to the fact that non-parabolic bands were considered in [5].

4.2 Velocity Field Characteristics

The characteristic relationship between electric field, $\vec{\mathcal{E}}$, and the steady state carrier velocity $\vec{v}$ in a semiconductor is an experimentally measurable quantity. Because of this fact, it is one of the most useful tests of a solid state Monte Carlo simulation. Since the $\vec{\mathcal{E}} - \vec{v}$ characteristics do not depend on the device being simulated, it is possible and desirable to divorce the simulation from all real space considerations. The simulation that is left is referred to as a $\vec{k}$-space or ensemble Monte Carlo simulation. The divorce from real space means that there is no tracking of positions, no boundary conditions, no contacts, and no solution to Poisson's equation. The $\vec{\mathcal{E}} - \vec{v}$ relationship is determined by subjecting every particle to the same electric field, allowing them to reach steady state, and then calculating the average velocity of the particles in the direction of the field. The results from the $\vec{k}$-space simulation are shown in figure 4.1. These simulated results compare favorably with the experimental results given in [12]. The noise in figure 4.1 is due to the fact that the ensemble average rather than the time ensemble average was used.

4.3 MESFET Description and Simulated Characteristics

The device simulated was a n-type GaAs MESFET as shown in figure 4.2. The geometric parameters of the device were chosen to match those used by [2]. The reader is reminded that under normal operating conditions the gate is negatively biased. The conducting channel will exist between the space charge region under

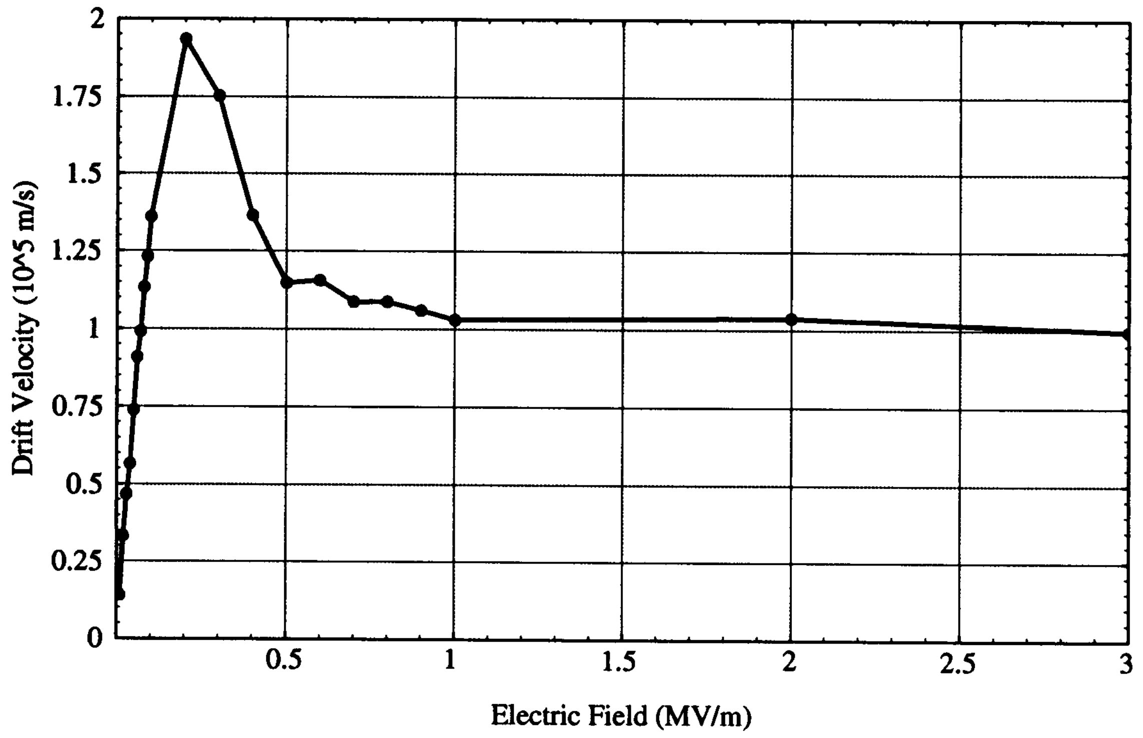


Figure 4.1: Simulated velocity field curve for GaAs.

the gate and the undoped substrate. The device simulated is actually intended to be a small portion of a much larger device. The larger device has a gate width of 200μm and the source and drain pads are 200μm square. The currents presented are scaled to represent the 200μm gate width. The simulation parameters are given in table 4.1.

The purpose of the simulations presented in this section is to verify that the MESFET exhibits reasonable characteristics. The carriers are initially distributed as shown in 4.4. At the start of the simulation a set of fixed potentials is applied to the contacts. The simulation is then allowed to run to steady state. Steady state is defined by equal and opposite source and drain currents. Figure 4.3 shows the time integral of each of the terminal currents for a typical set of applied voltages. Notice that steady state occurs after approximately 6ps. Figure 4.5 shows a typical spatial distribution of particles at steady state.

A number of bias points were simulated to determine their drain currents. The results were compiled into the set of characteristic $I_D - V_D$ curves shown in

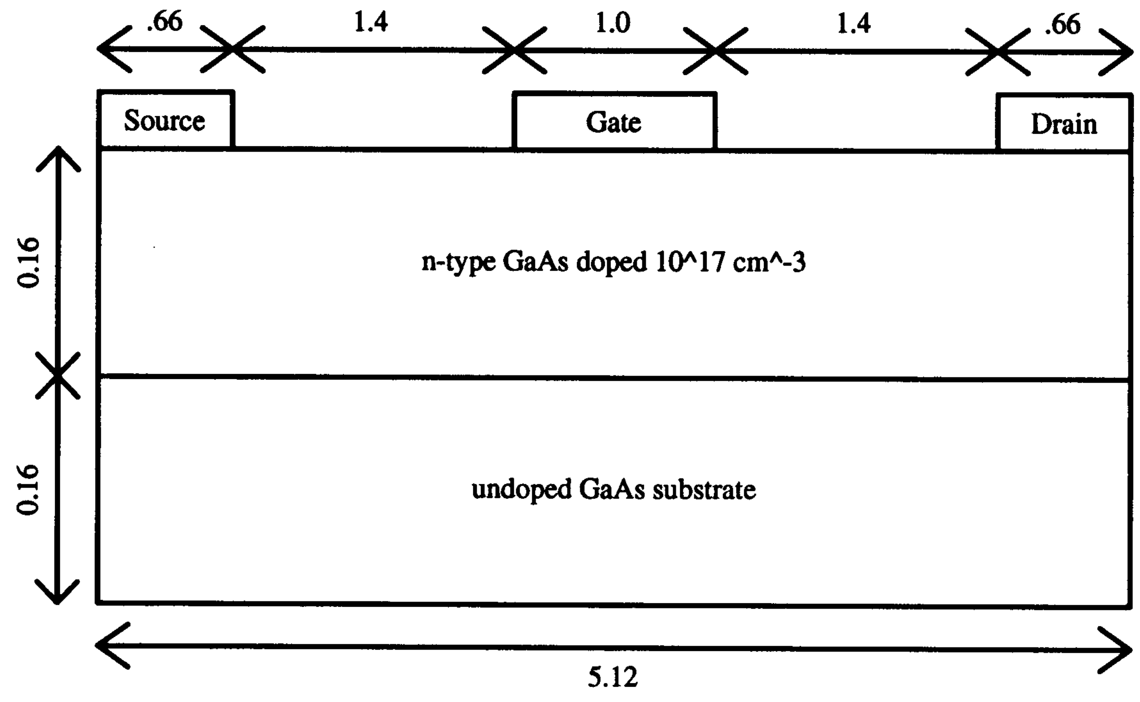


Figure 4.2: Cross-sectional view of simulated device, after [2]
Dimensions in μm, not to scale.

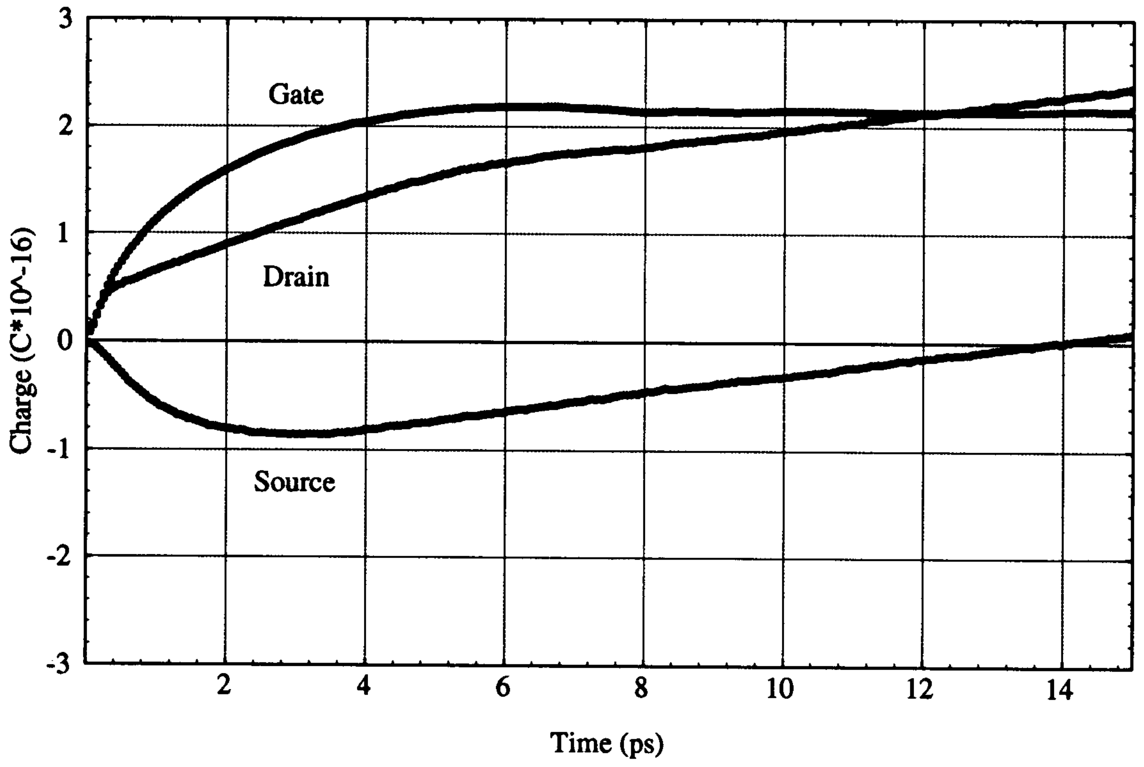


Figure 4.3: Integral of the current through each contact for a typical bias point.

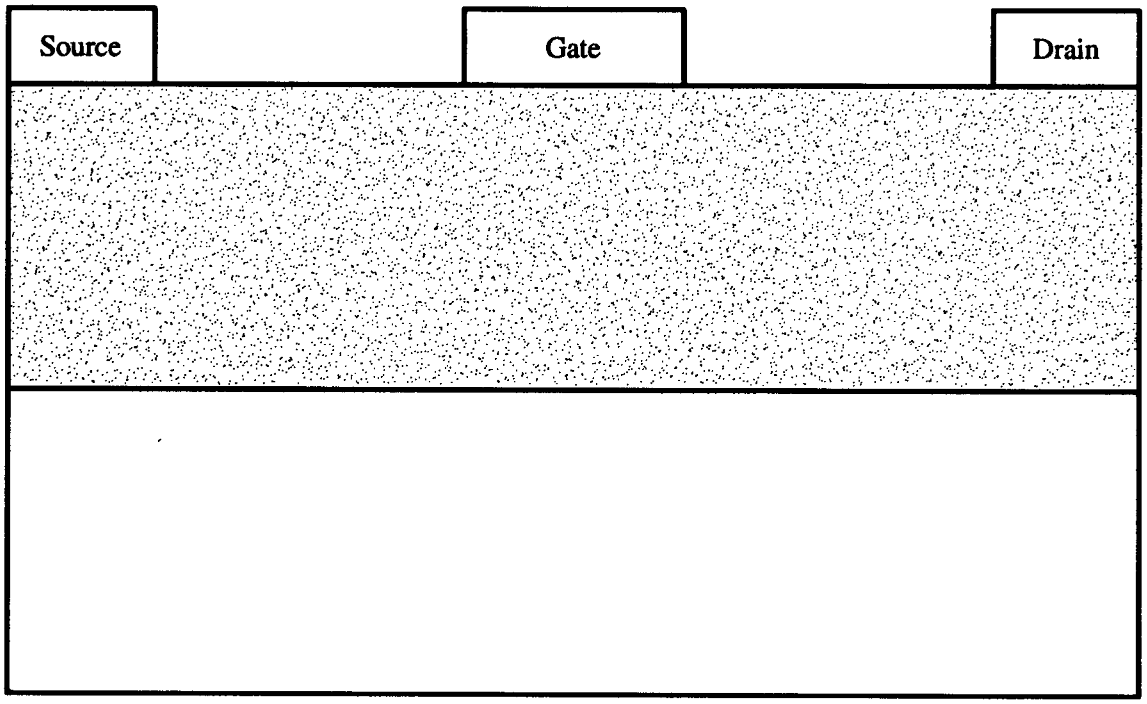


Figure 4.4: Typical initial spatial distribution of particles.

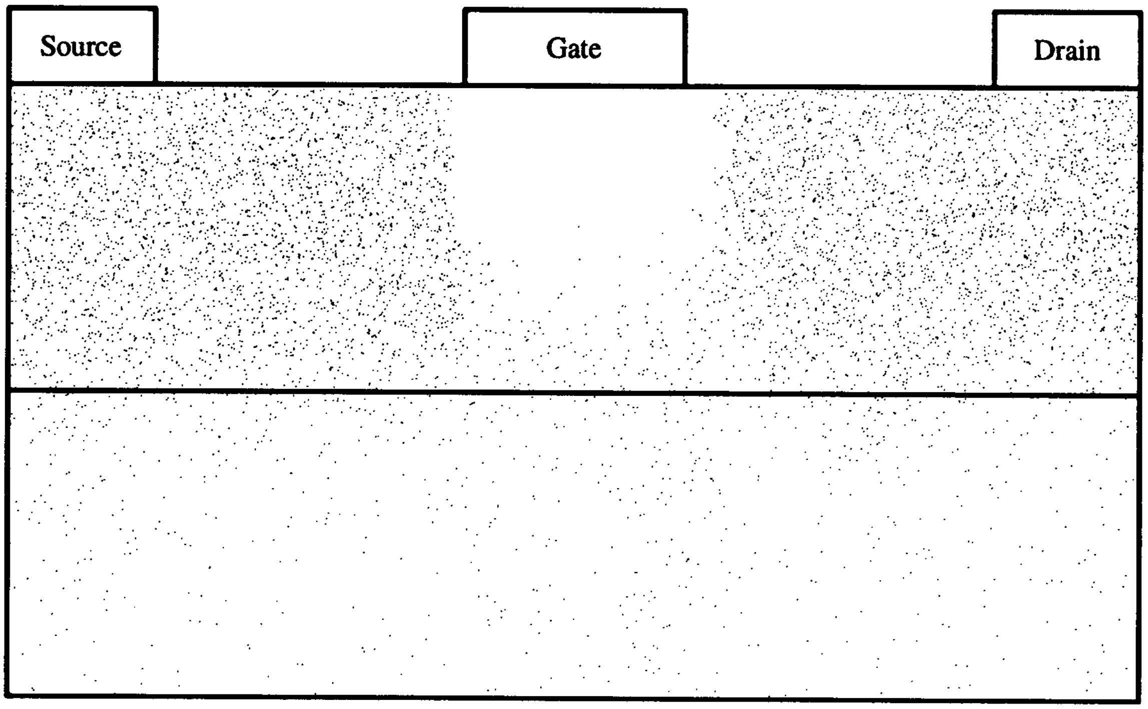


Figure 4.5: Typical final spatial distribution of particles.

Parameter Description	Parameter Value
Simulation Time Step	0.01ps
Time Steps Between Poisson Solutions	5 (0.05ps)
Initial Number of Particles	8000
Poisson Accuracy	0.001
Length of Simulation	15 ps
Lattice Temperature	300K
Maximum Energy Included in Scattering Rate Tables	3eV
Number of point in Scattering Rate Tables	600

Table 4.1: Simulation parameters.

figure 4.6 and the $I_D - V_G$ curve presented in figure 4.7. The set of bias points were chosen to match those used in [2]. Figure 4.6 can be compared with figure 10-15 in [2]. The $I_D - V_D$ curves have the expected form, but the currents are 20 to 30% smaller than those presented in [2]. This result is not particularly surprising or worrisome since [2] does not include descriptions of the scattering mechanisms or physical parameters used. These results are considered proof that the simulation is functioning correctly.

4.4 Speed Improvements with Parallelization

The speed improvement as a function of the number of processors used was measured. To assure reliable results, the system was run in single user mode with a minimum number of system processes active. The one processor case used the code compiled for serial execution. The code used did not exploit the vector capabilities of the computer. The same bias point was, of course, used for all timings. Figure 4.8 shows both the ideal speed increase and the observed speed increase. The deviation from the ideal case comes from two factors. First, as discussed in section 3.2, the Titan's bus will saturate as more processors are used. This effect represents most of

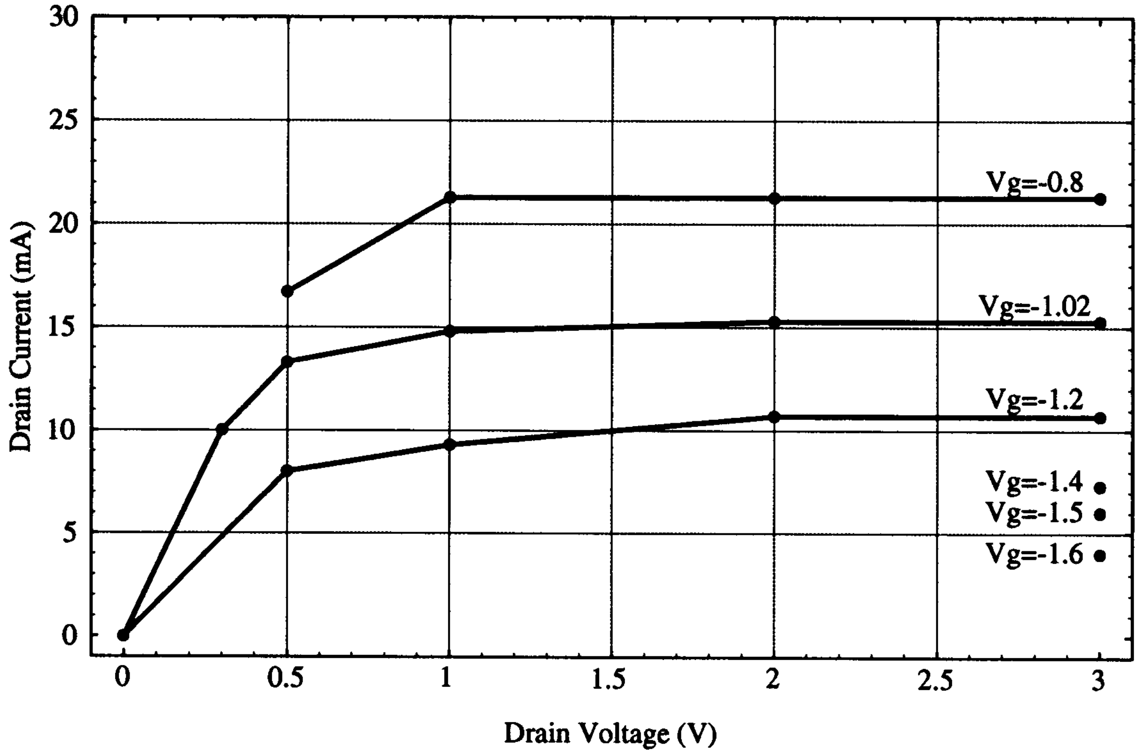


Figure 4.6: Drain current as a function of drain voltage for the simulated device.

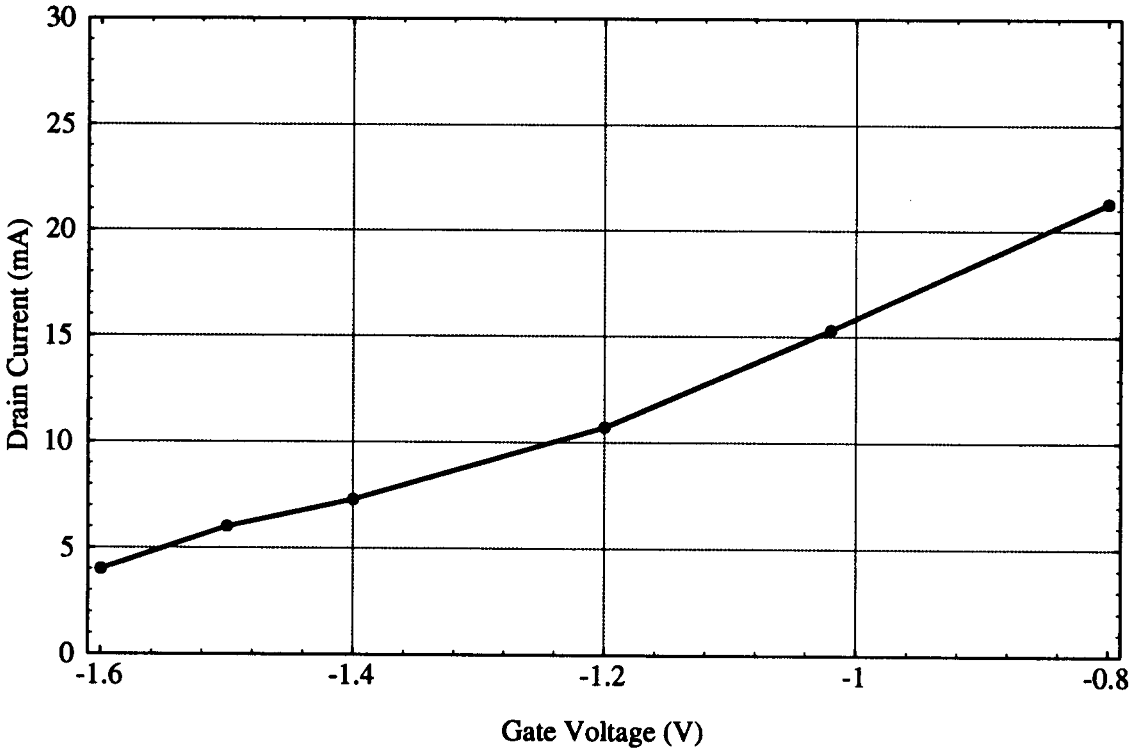


Figure 4.7: Drain current as a function of gate voltage for the simulated device, $V_D = 3.0V$.

the deviation from the ideal case. The second effect arises from those parts of the simulation which are not parallelized. As the number of processors increases, this effect becomes more important. However, with four processors, the serial portions of the program represent less than 1% of the total computer time. Hence, this effect is not considered significant.

4.5 Speed Improvements with Vectorization

The speed improvement using the vector units was also investigated. The measurement conditions were identical to those presented in section 4.4, except that the code was allowed to exploit the vector capabilities of the machine. The results are presented in figure 4.9 with the serial-vector case assigned to be unity. Table 4.2 gives the times for these simulations.

Comparing figures 4.8 and 4.9, it is seen that the use of the vector units saturates the bus more quickly. This behavior is exactly what is expected from the discussion presented in section 3.4. The specific fact that there is very little speed increase in going from two to four processors is also expected from the discussion in section 3.4.

Figure 4.10 shows both the vectorized and non-vectorized cases, normalized to the non-vectorized serial case. The reader should observe that the two curves converge to approximately the same point. This result lends support to the idea that the non-ideal behavior is due to bus limitations.

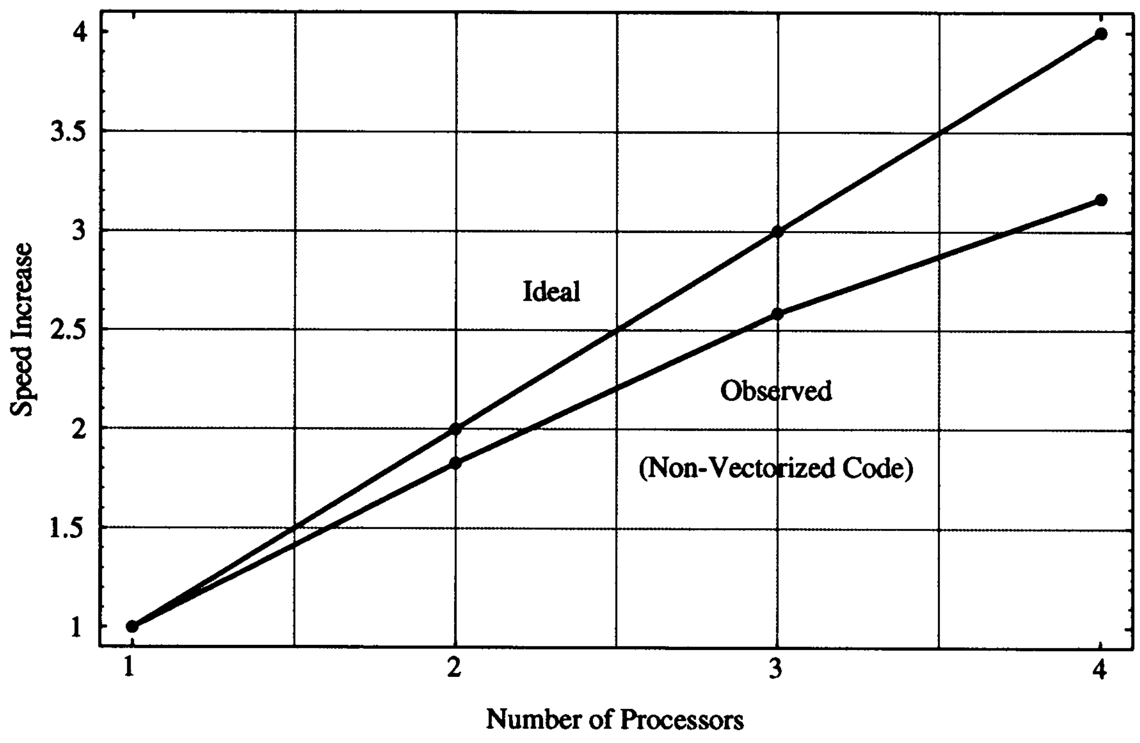


Figure 4.8: Simulation speed as a function of the number of processors used, for the non-vectorized case.

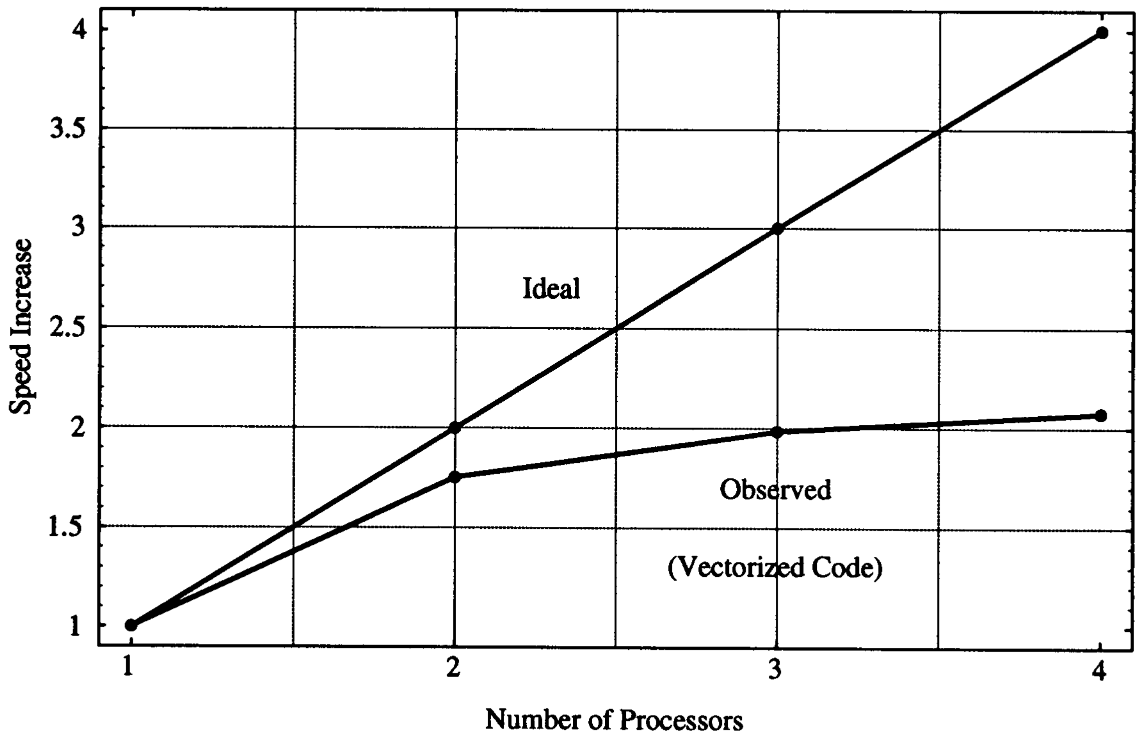


Figure 4.9: Simulation speed as a function of the number of processors used, for the vectorized case.

# Proc.	Non-Vectorized (Hours)	Vectorized (Hours)
1	21.2	13.5
2	11.6	7.7
3	8.2	6.8
4	6.7	6.5

Table 4.2: Simulation times.

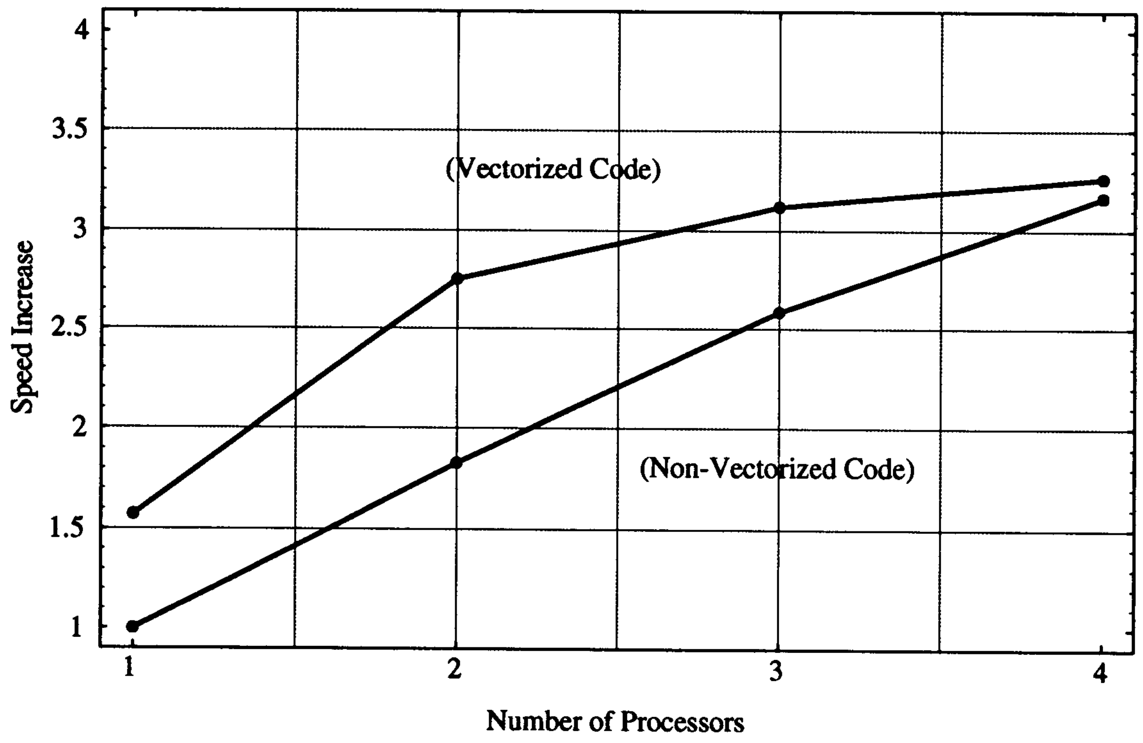


Figure 4.10: Comparison of the speed increase for the vectorized and non-vectorized cases.

Chapter 5

Conclusions and Recommendations

5.1 Conclusions Regarding Parallelization

The speed increase achieved with parallelization on the Titan II is primarily limited by memory access. This conclusion is based on two observations. First, both the vectorized and non-vectorized cases exhibit approximately the same speed increase for the four processor case. Second, when the simulation is being executed, the status lights on Titan II indicate an increased number of memory access collisions. Considering that contemporary single processor workstations are beginning to exhibit memory access limitations, it is unlikely this problem will be adequately and inexpensively resolved in the near future. This limitation will at least limit the number of processors in shared memory machines in the forseeable future.

Parallel programming on the Titan is a difficult and time consuming task largely due to the lack of good development and debugging tools. Due to this fact, care must be taken in deciding whether or not to parallelize future programs. Specifically, there are four points that should be given close consideration when making the decision. First, how much will the program be used. Second, will the program be used in a manner where the results from one simulation will determine the input parameters to the next. Third, what is the current state of the parallel debugging tools. Finally, what speed increase can be expected from the given computer.

5.2 Conclusions Regarding Vectorization

When vectorization is used with parallelization on the Titan II, there is very little speed increase from it. However, when used alone, vectorization can provide a significant speed increase. As a reminder to the reader, it is noted again here that only the potential solution portion of the program was vectorized. An additional speed increase factor would be gained if the particle movement could be vectorized.

The time and effort required to achieve vectorization in the Poisson solver was substantial, but it was not nearly as tedious as parallelizing both the Poisson solver and particle movement. The vector debugging facilities on the Titan II function considerably more reliably than the parallel debugging facilities. For the specific case of the Titan II, vectorization provided a much better speed increase per day of development time than parallelization. In general, the decision to vectorize a program should be based on four criteria. First, how much will the program be used. Second, what speed increase, if any, can be expected from vectorization. Third, what is the current state of the vector debugging facilities. Finally, will parallelization defeat any benefit from vectorization.

5.3 Recommendations for Future Work

Several ideas for improving the program should be presented here. First, the code could be modified to use previous final states at the initial state for later runs. This could significantly reduce the time required for the simulation to reach steady state. Second, vectorization could be extended to include the particle movement portion of the program. Third, in some cases it may be possible to avoid the effort required to parallelize code by simply running multiple copies of the program. Finally, it may be possible to implement multitasking. Multitasking refers to using processors in parallel to perform different tasks. For example, since the fourth processor does not provide much of a speed increase in the vectorized case, it might be possible to use that processor to relieve the other three processors of a task that does not require much memory access. One possibility for this would be to use the fourth processor to generate all the random numbers for the simulation.

A few comments regarding this Monte Carlo simulation and such simulations in general may be useful. The first two comments are particularly important. First, the simulation implemented by Udaya Ranawake [9] on a 64 node N-Cube computer shows considerably more promise for expansion of the simulation. Second, the development of advanced, reliable development tools on a particular parallel computer should have a substantial influence on whether that machine is used for parallel program development. Finally, regarding the simulation presented here, maintenance of the parallel code when small modifications are made is probably a good use of time. However, if major modifications are made, the decision to maintain the parallel code should be made with great care.

Bibliography

[1] Jenifer Lary. Ensemble monte carlo simulation. EE519 Term Project for Dr. Steve Goodnick.

[2] Roger W. Hockney and James W. Eastwood. *Computer Simulation Using Particles*. McGraw-Hill, 1981.

[3] Carlo Jacoboni and Paolo Lugli. *The Monte Carlo Method for Semiconductor Device Simulation*. Springer-Verlag, 1989.

[4] Ronald M. Yorston. Free-Flight Time Generation in the Monte Carlo Simulation of Carrier Transport in Semiconductors. *Journal of Computational Physics*, 64:177–194, 1986.

[5] Mark Lundstrom. *Fundamentals of Carrier Transport*. Addison-Wesley, 1990.

[6] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[7] Kiyoyuki Yokoyama Masakki Tomizawa and Akira Yoshii. Nonstationary Carrier Dynamics in Quarter Micron Si MOSFET's. *IEEE Transactions on Computer-Aided Design*, 7(2), 1988.

[8] Chang G. Hwang Doreen Y. Cheng and Robert W. Dutton. PISCES-MC: A Multiwindow, Multimethod 2-D Device Simulator. *IEEE Transactions on Computer-Aided Design*, 7(9), 1988.

[9] Udaya Ranawake. *Cluster Partitioning Approaches to Parallel Monte Carlo Simulation on Multiprocessors*. PhD thesis, Oregon State University, 1992.

[10] Stardent Computer Inc. *Titan II Programmer's Guide.* Part # 340-0116-01; Stardent Computer Inc. is now Kubota Pacific Computer.

[11] Stardent Computer Inc. Course Notes Regarding Titan II Parallel Programming. A fax provided by Geoff Mater at Stardent.

[12] S. M. Sze. *Physics of Semiconductor Devices.* Wiley, 1981.