Parallel and Distributed Three-Dimensional Monte Carlo Semiconductor Device Simulation

Henry Sheng, Roberto Guerrieri†, and Alberto Sangiovanni-Vincentelli

Technical Report ERL-95-52
Department of Electrical Engineering and Computer Sciences
hsheng@ic.EECS.Berkeley.EDU
http://www-cad.eecs.berkeley.edu/~hsheng

† Dipartimento di Elettronica e Informatica
Università di Bologna, Italy

Abstract

We present a parallel Monte Carlo algorithm for the simulation of semiconductor devices in three dimensions. The physical behavior of the system is governed by the Boltzmann Transport Equation. In the absence of direct interactions among charge carriers, the samplings of the statistical space are independent. This results in a potential decoupling of computational tasks that reduces the number of required communications in simulation. A Monte Carlo algorithm, with both static and dynamic load balancing strategies, is shown for the Connection Machine CM-5 and a network of workstations. We show that both load balancing strategies scale well with problem size, and present results on relative speedups between them. A randomized analysis of performance is presented and physical simulation results are shown.

1 Introduction

The push for higher integration densities in semiconductor device design and the consequent reduction of minimum feature sizes have placed tremendous demands on the capabilities of today's simulation tools. High order effects, such as hot electrons, velocity overshoot, and ballistic transport, become significant for deep-submicron devices. Monte Carlo device simulation [1, 2], based on a microscopic treatment of electron behavior in semiconductors, offers a solution which is becoming increasingly important as microelectronic process technology now focuses on the deep-submicron ULSI (Ultra Large Scale Integration) regime, where feature sizes are less than 0.25 microns. Unfortunately, the large computational requirements of Monte Carlo simulation, which may be on the order of days of CPU time for a typical MOS transistor simulation, prohibits its direct use in practical device engineering.

Despite variance-reduction algorithms, such as particle multiplication and importance sampling [3, 4, 5, 6], for decreasing the CPU time of Monte Carlo solutions, current simulators are primarily focused on the two-dimensional Monte Carlo problem. In [7] and [8], vector approaches are pursued on the IBM 3090/600E and the Cray XMP, while in [9] a parallel approach to the Monte Carlo problem is pursued on the Intel iPSC/860 hypercube using 32 processors. While the direct comparison of relative performance is difficult because there is no consistent metric of computational efficiency [10], it is clear that the computational requirements of practical Monte Carlo simulation are beyond the abilities of these machines. For instance, in [7], a self-consistent Monte Carlo simulation of a two-dimensional silicon MOSFET required several days of CPU time. In [8], two-dimensional MOSFET simulation required 45 to 90 minutes of CPU time for a single bias condition. The extension of Monte Carlo simulation to full three-dimensional problems requires large-scale parallelism to provide the necessary computational power and scalable algorithms to take advantage of the parallelism. In our previous work, a data-parallel Monte Carlo device simulator in three-dimensions [11, 12, 13] was developed for the Connection Machine CM-2.

In this paper, we present a scalable coarse-grained Monte Carlo algorithm for both the Connection Machine CM-5 and a network of workstations using the Parallel Virtual Machine Tool (PVM) [14]. This algorithm solves the three-dimensional problem for a single-carrier static-field simulation. The amount of computation involved is non-deterministic, since the number of simulated electron free flights is randomly distributed. Static and dynamic load balancing strategies are developed to address this issue, and the resulting scalabilities and speedups are shown. This paper is organized as follows: The Monte Carlo device simulation problem is introduced in Section 2. Section 3 provides an overview of the simulation algorithm. Section 4 discusses the static and dynamic load balancing algorithms. Section 5 discusses the computational results, including scalability and load balancing, and physical results from the simulation of device structures.

2 Problem Definition

Device simulation predicts the behavior of charge carriers in semiconductor materials. This simulator addresses the problem of static-field Monte Carlo for the study of stationary processes [11]. The simulator follows a single-*carrier* (electron) treatment, which is accurate for structures such as n-MOSFET's where electron behavior is dominant. A single-*particle* Monte Carlo paradigm [2] is taken, where the trajectory of a single charge carrier is followed in time. By simulating a sufficiently long time period, enough sampling is performed over the domain space so that reliable Monte Carlo estimates can be formed.

Formally, the solution to the Boltzmann Transport Equation [15] is sought (seen in equation 1), compliant with Poisson's Equation (equation 2):

$$\frac{\partial f}{\partial t} = -\vec{v} \cdot \nabla_{\vec{r}} f - \vec{F} \cdot \nabla_{\vec{p}} f + \left. \frac{\partial f}{\partial t} \right|_{coll} \tag{1}$$

$$\nabla \cdot (\epsilon \nabla \varphi) = \rho \tag{2}$$

Here f, $\vec{r}$, and $\vec{p}$ are the distribution function, position vector, and momentum vector, respectively, and φ is the potential and ρ is the net charge concentration. The last term of the BTE is the collision term, and is responsible for scattering and generation/recombination of carriers. Poisson's equation resolves the incident forces experienced by the particles. This simulation is semi-classical, so that the kinematics of carrier free flight are treated through classical physics, while the dynamics of carrier scatterings are treated quantum-mechanically.

A spherical non-parabolic conduction band is used. The deviation from parabolic behavior is expressed with a non-parabolicity parameter α [2]. The carrier energies and the kinematics of carrier flights are governed by:

$$E(1 + \alpha E) = \frac{\hbar^2 \vec{k}^2}{2m^*} \tag{3}$$

Here, E is the carrier energy, $\vec{k}$ is the carrier wave vector, m^* is the conductivity effective mass, and $\alpha = 0.5$.

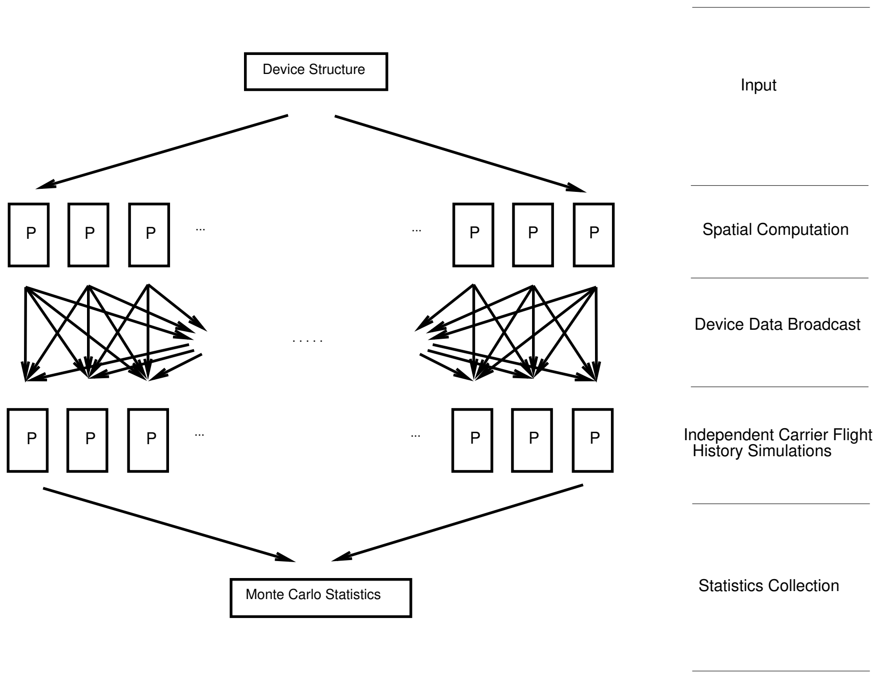

Figure 1: High-level parallel Monte Carlo simulation algorithm.

We have included the scattering mechanisms used in [16]: acoustic phonons, optical phonon emission and absorption, ionized impurity scattering, impact ionization, and surface roughness scattering. In the case of impact ionization, the effects of variable particle populations is assumed to be negligible. As in [16], the simulator is a post-processor to moment-based solutions, which are used both for the processing of distribution-based phenomena such as carrier shielding of potentials and for the calculation of electric fields in the device. These solutions are provided by a three-dimensional drift-diffusion simulator running on the Connection Machine [17].

3 Parallel Algorithm

The Monte Carlo algorithm involves the simulation of a large number of statistically independent flight histories, which begin with the injection of a carrier into the device and terminate when the carrier exits. The limit on the actual number of flight histories can be determined either *a priori* or dynamically during the simulation based on the standard error of the Monte Carlo estimates. By ergodicity, the simulation of n injections of a single particle into a device (with each injection occurring after the particle exits the device) is equivalent to the concurrent simulation of n particles which are injected simultaneously and treated independently. Thus, although many carrier flights are simulated in parallel, the simulator conforms to a single-particle classification. The parallel algorithm (Figure 1) is composed of three main steps: spatial computation and broadcast, carrier flight simulation, and statistics collection.

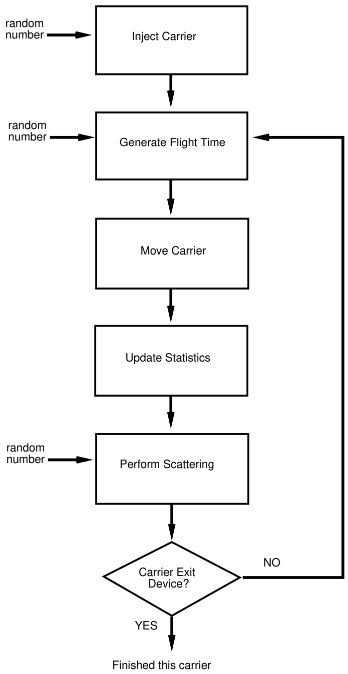

Figure 2: Flight history simulation algorithm for a single carrier.

3.1 Spatial Computation and Broadcast

The first step of the computational problem generates the spatial data structures of the semiconductor device. The spatial structure of the device is discretized into a rectangular grid. For each grid point, the electric fields and a table of scattering rates (Γ values, as will be discussed in the section on flight times) must be computed. Even for moderately sized grids, this task can be demanding (a 64x64x64 spatial discretization yields 256K grid points). This task was parallelized, so that each processor computes the required values for a subsection of the entire grid. The grid points are partitioned so that the load imbalance requires the processing of at most one extra grid point for any processor. Since there are no communications required, the only constraint on partitioning ordering is load balance. The results of these calculations are globally broadcasted.

3.2 Carrier Flight Simulation

The simulation of charge carriers involves the iterative simulation of carrier flights. Because the carriers are statistical wave packets, the free flight times and scatterings are treated statistically. For each processor, the algorithm for single-particle Monte Carlo simulation is outlined in Figure 2. After the conclusion of flight histories, new ones are begun if the limit has not been reached. The processing of carrier flights involves five main steps:

3.2.1 Carrier Injection

The first step is the injection of a carrier into the device. The carriers are thermally distributed and injected into ohmic contact regions with random energies according to a Maxwellian-Boltzmann distribution at $T = 300^o K$.

3.2.2 Flight Time Generation

The free flight duration of electrons is determined by the scattering rate $\gamma(\vec{r}, \vec{k})$, where $\vec{r}$ is the position of the carrier within the device and $\vec{k}$ is its wave vector. $\gamma(\vec{r}, \vec{k})$ is defined to be the summation of probabilities over all scattering mechanisms:

$$\gamma(\vec{r}, \vec{k}) = \sum_{i=1}^{n} \gamma_i(\vec{r}, \vec{k}) \tag{4}$$

where $\gamma_i(\vec{r}, \vec{k})$ is the scattering probability, associated with scattering mechanism i, of a carrier with position $\vec{r}$ and wave vector $\vec{k}$. Suppose $\gamma(\vec{k}(t), \vec{r}(t))dt$ is the probability of a scattering event during some time interval dt for a particle. The probability that this particle will suffer a collision in time period $[t, t + dt]$, call it $\mathcal{P}$, is given by:

$$\mathcal{P}(t)dt = \gamma(\vec{k}(t), \vec{r}(t)) \cdot \exp\left[-\int_0^t \gamma(\vec{k}(t), \vec{r}(t))d\tau\right] dt \tag{5}$$

The inversion of the cumulative distribution function (cdf) associated with this probability will lead to the selection of a flight time from a random variate evenly distributed in the interval $[0, 1]$:

$$\int_0^{t_f} \mathcal{P}(\tau)d\tau = 1 - r \tag{6}$$

Here r is a random number on $[0, 1]$. The flight time of the carrier, t_f, can be found through the inversion of this integral. This inversion is difficult and must be performed repeatedly during a simulation. Since the numerical evaluation of the integral is too slow for practical purposes, the technique used to compute the flight time is based on a self-scattering [2, 18] or variable Γ [16, 19, 20] approach. A fictitious scattering mechanism (self-scattering) is introduced so that the total scattering probability, Γ, is a piecewise constant function under the condition that $\Gamma \geq P(\vec{k}(t), \vec{r}(t)), \forall \vec{k}, \vec{r}$. Because this new scattering mechanism is fictitious, carrier states remain unchanged if the flight is terminated by a self-scattering and the algorithm does not interfere with the correctness of the physical result. Thus, if $\gamma(\vec{k}(t), \vec{r}(t))$ is replaced with a constant Γ, then:

$$\mathcal{P}(t)dt = \Gamma \exp^{-t\Gamma} dt \tag{7}$$

By replacing the $\mathcal{P}$(t)dt term, the flight time can be found with the selection of a random number, r, on [0,1]:

$$t_f = -\frac{1}{\Gamma} \ln(r) \tag{8}$$

3.2.3 Carrier Movement

Once the duration of free flight is determined, the state $(\vec{r}, \vec{k}, t)$ of the carrier is updated. t is known from the free flight time, while $\vec{k}$ and $\vec{r}$ are computed from the analytical integration of force and velocity over time, respectively.

$$\vec{k}_{final} = \vec{k}_{initial} + \frac{\vec{F}}{\hbar} t \tag{9}$$

$$\vec{r}_{final} = \vec{r}_{initial} + \int_0^t \frac{\hbar \vec{k}(t')}{m(1 + 2\alpha E(t'))} dt' \tag{10}$$

In equation 9, $\vec{F}$ is the force experienced by a carrier under the assumption of a piecewise constant discretization of the electric field.

3.2.4 Statistics Update

In this phase of the algorithm, statistical sampling is performed on the carriers [2]. The carrier residence in a particular state $(\vec{r}, \vec{k})$ is proportional to the value of the distribution function at that point. Statistics are only sampled in the case that the carrier experiences a true scattering.

3.2.5 Carrier Scattering

The carrier must finally be scattered. A scattering mechanism, responsible for terminating the carrier free flight, is chosen at random from a distribution where each mechanism from equation 4 occurs with probability $\frac{\gamma_i(E)}{\Gamma}$, and self-scattering occurs with probability $\frac{\Gamma - \gamma(E)}{\Gamma}$. If the mechanism responsible for terminating a carrier flight is a self-scattering event, then the carrier state is preserved. The probability distributions for scattering angles differ according to the causal mechanism. Anisotropic deflection angles are used for ionized impurity scattering. In surface roughness scattering, the deflection angle in the plane parallel to the $Si - SiO_2$ interface is chosen at random. Isotropic behavior is assumed for the remaining mechanisms. In addition, for the case of optical phonon absorption or emission, a discrete energy (the optical phonon energy) is gained or lost, respectively.

3.3 Statistics Collection

The remaining computational problem consists of the global compilation of statistical results among the individual processors. Each processor has local statistical information on particle simulation. This last step involves additive reduction operations among all processors for each statistical estimator and normalization value.

4 Load Balance

4.1 Static Load Balancing

The first approach in developing a strategy for load balancing was to perform a separate single-particle Monte Carlo simulation, consisting of m flight histories, on each processor. For n processors, mn total flight histories are simulated. The CPU time required to simulate each flight history is not deterministic and is distributed according to an unknown probability distribution. An empirical distribution is shown in Figure 3 (for the n^+nn^+ device structure in Figure 4). Because of the strong peak, the variance of this distribution is qualitatively small. For most practical devices, strongly-peaked distributions arise because of the existence of ohmic contact regions. In these cases, the load balance is expected to be reasonable for large m, because the load on each processor is independently and identically distributed with small variance. For this scheme, the total CPU time required for the simulation is equal to the maximum CPU time among all processors required to simulate m flight histories.

4.2 Dynamic Load Balancing

The static load balancing approach is expected to work well in the case of large m. However, the underlying probability distribution for the CPU time required to simulate a flight history in a particular device is

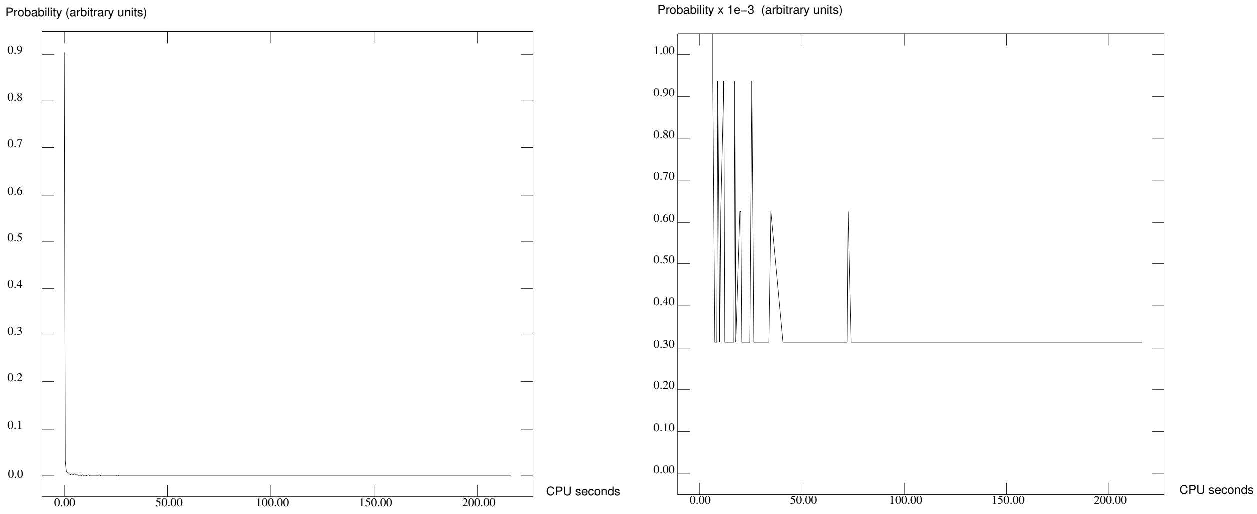

Figure 3: Empirical distribution function of CPU time required for one flight history. The strongly-peaked nature of this distribution implies small variance. The right side graph is a magnification of the left.

unknown, and is not guaranteed to be strongly-peaked. This distribution is dependent on the structure and bias of the target device. In addition, there is no guarantee that load balance will be achieved, since the magnitude of m required for statistical confidence in load balance may not be known *a priori*. This algorithm is particularly ill-suited for workstation clusters where the processor loads are unpredictable.

An alternative approach was developed with dynamic load balancing. One process (the particle manager) is responsible for maintaining a centralized event queue for managing the flight histories. When a processor is ready to simulate a new flight history, a request is sent to the particle manager. The particle manager processes this request, based on the number of flight histories remaining in the event queue. After a processor receives a rejection of its particle request, the time that it must lie idle (hence, the maximum load imbalance) before the end of the simulation is, at most, the time required for simulating one flight history.

For the CM-5, the particle manager simulates its own flight histories in addition to appropriating tasks. In this case, the latencies for the requests and replies are significant, since the simulation speed of the particle manager is impacted by latencies from communicating with all processors. These latencies are decreased through interrupt-driven active messages [21, 22]. Active messages are well suited for the asynchronous demands of the load management algorithm, since the message latencies are partially hidden and the messages are non-blocking.

For workstation clusters, the particle manager is spawned as a separate process from the actual simulation engines. The simulation engine(s) which resides on the same physical processor as the particle manager is minimally affected, since the particle manager requires only 184Kb of memory, and spends the majority of time waiting for requests.

5 Performance and Physical Results

For our experiments, the CM-5 version of the Monte Carlo simulator used CMMD version 3.0 for the message passing and was run on a 32 node machine. The network of workstations include 4 ALPHA-based machines, an ALPHA-MP machine, 60 MIPS-based machines, and a 486DX2/66. The target device structure

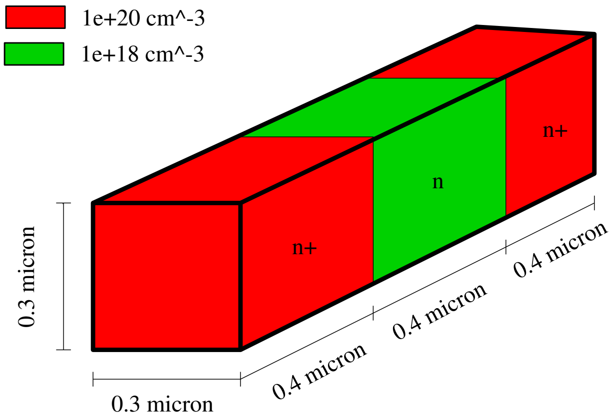

Figure 4: n^+nn^+ device structure.

used for obtaining results for load balance and scalability was a psuedo-1D n^+nn^+ device (Figure 4), with an applied bias of 3 volts. A $2 \times 128 \times 2$ grid was used for the spatial discretization.

5.1 Load Balance

The n^+nn^+ device structure was simulated on the CM-5 using 3200 total flight histories with both static and dynamic load balancing. Because the processor workloads of the CM-5 are predictable, this architecture allows an accurate analysis of load balance. We can consider each individual flight history simulation to be a task with a non-deterministic cost (CPU time) which can be considered as a random variable. The goal of the load balancing algorithms is to distribute these tasks among the processors so as to minimize the difference between the largest and smallest total processor cost. The quality of the load balancing algorithms can be seen by observing histograms of the resulting processor loads. Histograms of individual active CPU times (not counting wait cycles at synchronization boundaries at the end of simulation) for both static and dynamic load balancing schemes are shown in Figure 5. The distribution of CPU times for the dynamic algorithm yields a sharper distribution, so that fewer cycles are wasted waiting for the last processor to finish. The total CPU time required by the simulator is the maximum of the individual processor CPU times. In this example, dynamic load balancing yields a 37.4% speedup over the static method.

For workstation clusters, only the dynamic load balancing algorithm is useful, because the workloads of the machines are not well behaved. With static load balancing, the run-time of the algorithm is dictated solely by the slowest node (either because of processor speed or because of processor workload).

5.2 Scalability

5.2.1 Problem Size Scalability

Both the static and dynamic load balancing algorithms were used to analyze problem-size scalability of the simulator for the CM-5 and a cluster of 48 DECstation 5000/125's. Both algorithms scale approximately linearly with the problem size (Figure 6). Hence, they can be fit to: $y = mx$. The results from the workstations exhibit more unpredictability because of the uncertainty caused by other workloads. An implication of linearity is that both load balancing strategies result in efficient algorithms in terms of scalability.

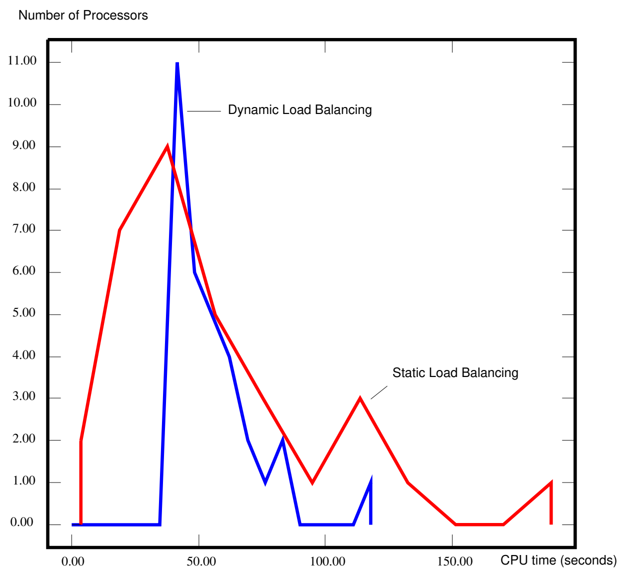

Figure 5: Histogram of individual processor active CPU times for static and dynamic load balancing. The total CPU time is the maximum processor CPU time.

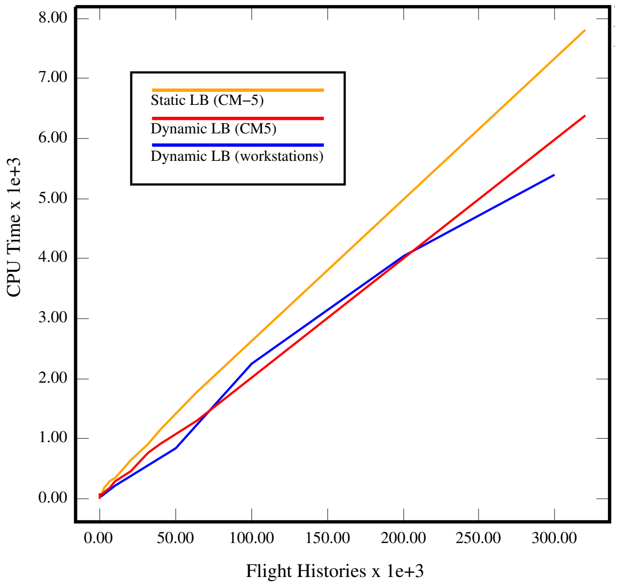

Figure 6: Effects on problem size scaling with run-time. The problem size is defined to be the number of flight histories simulated.

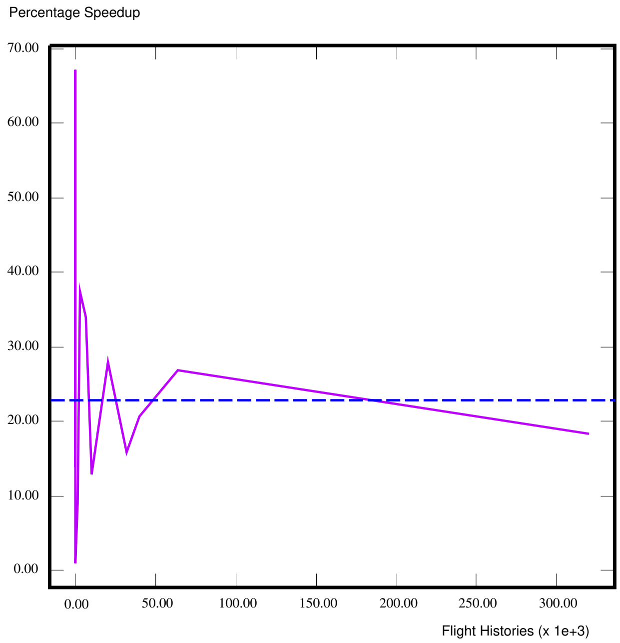

Figure 7: Speedup of dynamic load balancing over static load balance versus total number of flight histories simulated. The dotted line is the expected speedup, $\frac{\hat{m}_{static}}{\hat{m}_{dynamic}} = 0.2232$.

In the case of dynamic load balancing, the expected slope, $m_{dynamic}$, is T_{fh}, the expected CPU time required for the simulation of a flight history. This value is heavily dependent on the nature of the target device structure. In the case of static load balancing, the expected slope, m_{static}, is a more complicated value. We define A to be the average time required to simulate a flight history in a particular processor. Each A will be randomly distributed according to some distribution whose expectation will be T_{fh}. The value of m_{static} is the expected value of the maximum A among all processors.

$$m_{dynamic} = T_{fh} = E\left[A\right] \tag{11}$$

$$m_{static} = E[maxA_i] \quad i = 1..n \tag{12}$$

Since $E[maxA_i] \geq E[A]$, it follows that $m_{static} \geq m_{dynamic}$. For the CM-5 results on the n^+nn^+ structure, the empirical estimates $\hat{m}_{static} = 0.02440$ and $\hat{m}_{dynamic} = 0.01995$ were obtained.

The speedup of the dynamic load balancing strategy over the static load balancing scheme is shown in Figure 7. The speedup is centered around S, which is related to the ratio of the slopes in Figure 6:

$$S \approx \frac{m_{static}}{m_{dynamic}} - 1 \tag{13}$$

The expected speedup is therefore a constant (for $\hat{m}_{static}$ and $\hat{m}_{dynamic}$, $\hat{S} = 0.2232$). For smaller problem sizes, the sample population of flight histories is small so that the variance associated with the speedup is large. As the problem size increases, the associated variance decreases accordingly, and the speedup converges towards S.

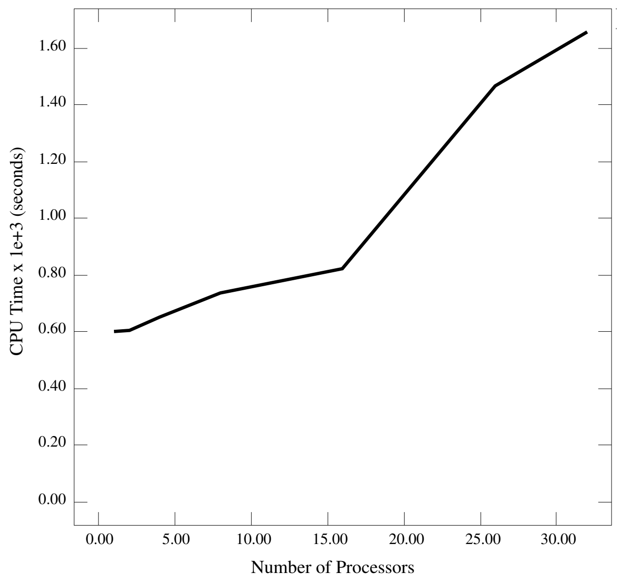

Figure 8: Architectural scaling with constant problem size per processor (1000 flight histories) on a DECstation 5000/125 network.

5.2.2 Architectural Scalability

For the CM-5, the static algorithm is expected to scale well with machine size (constant problem size per processor), since the only communications are performed at the beginning and end of simulation - in the initialization of the device structure and statistics compilation steps, respectively. However, load balance should become worse with machine size, since we expect the maximum A to be greater with larger machines sizes (because there are more candidate A values), and resultingly, we expect m_{static} to be dependent on the number of processors. As a result, we expect the run-time to grow slowly with the number of processors when scaling the machine size and keeping the problem size per processor constant.

The approximately constant behavior of the expected speedup of the dynamic load balancing algorithm over the static algorithm implies that the overhead of communications in the dynamic case is roughly constant (per flight history). This cost arises from the latency of the request operation at the beginning of a flight history. The roughly constant nature of this cost implies that very few collisions occur during communications. As a result, the run-time behavior of this simulator, with dynamic load balancing, scales well with the size of the machine (again, with constant problem size per processor). Furthermore, since m_{static} is dependent on the number of processors, we expect the speedup of the dynamic algorithm to improve with machine size.

For the network of workstations, a homogeneous cluster (DECstation 5000/125) was used to test architectural scalability (with constant problem size per processor). In this case, 1000 flight histories were simulated on each processor. Although the CPU time per flight history is non-deterministic, the variance, as seen earlier, is small. The algorithm scales reasonably with problem size (Figure 8). However, with large numbers of processors, contention for the particle manager occurs and runtime increases. The latencies of processor communications in this architecture is much greater than for the CM-5. Hence, bottlenecks at the particle manager occur which are sensitive to the number of processors. Currently, we are investigating statistically-based dynamic scheduling algorithms which control, during the course of simulation, the number of flight histories requested or granted per communication. The goal is to decrease the number of required

Device	Architecture	Time (sec.)
n^+nn^+ (2x128x2)	Workstations	406.2
	CM-5 (dynamic)	594.7
	CM-5 (static)	728.9
2D MOSFET (64x26x2)	Workstations	469.4
	CM-5 (dynamic)	735.2
	CM-5 (static)	807.7
3D MOSFET (64x26x4)	Workstations	502.6
	CM-5 (dynamic)	759.7
	CM-5 (static)	936.3

Table 1: Simulation times for 32,000 flight histories using dynamic and static load balancing approaches on the CM-5 and the dynamic approach on a cluster of 60 MIPS-based workstations. The three devices simulated are a n^+nn^+ device, a 2-dimensional MOSFET, and a 3-dimensional MOSFET structure. The associated grid sizes are shown.

communications while still guaranteeing load balance, with high probability, at the end of simulation.

5.3 Simulation Performance

The simulator was benchmarked on both parallel and distributed architectures (using both static and dynamic load balancing algorithms for the CM-5), as seen in Table 1. The workstation cluster consisted of 60 MIPS-based DECstations. The three devices simulated were a n^+nn^+ device (with a grid size of 2x128x2), a 2-dimensional MOSFET (with a grid size of 64x26x2), and a 3-dimensional MOSFET (with a grid size of 64x26x4). Each simulation consisted of 32,000 flight histories. The cluster of workstations outperformed the CM-5 by an average of 34% (geometric mean of the workstation to to CM-5 execution time ratio is 66%), while the dynamic load balancing algorithm outperformed the static algorithm by an average of 15.4% (geometric mean of the ratio between the time for the dynamic algorithm to the time for the static method is 84%). Scattering processing rates of 550,000 scatterings per second were achieved on the network of workstations and 364,000 scatterings per second on the CM-5. These are significantly faster than the Cray XMP [8] (15,000-30,000 scatterings per second for typical MOSFET applications) and the Connection Machine CM-2 [11] (200,000 scatterings per second).

5.4 Physical Results

The n^+nn^+ device structure was simulated using a total of 320,000 flight histories. For the CM-5, this simulation required 7806.9 seconds of CPU time with the static load balancing scheme and 6382.5 seconds of CPU time with the dynamic load balancing scheme. The static load balancing method is uninformative for the network of workstations, so only the dynamic load balancing, which required 5762.7 seconds, was used. The resulting distribution of electrons in the device can be seen in Figure 9. In Figure 10, the average velocity of carriers in the device can be seen as a function of position, parameterized by a set of applied biases. Velocity saturation is observed for high applied biases at $1.2 \times 10^5 m/s$. In addition, for a 5V bias, the onset of some non-local velocity overshoot effects can be observed at one $n^+ - n$ interface ($x = 0.45\mu$), where the carrier velocities are not at equilibrium with the local fields. Figure 11 shows the corresponding electron energy distributions in the n "channel" of the device for the same applied biases. In the "channel", the energy distributions are significantly different from the Maxwell-Boltzmann distribution that macroscopic

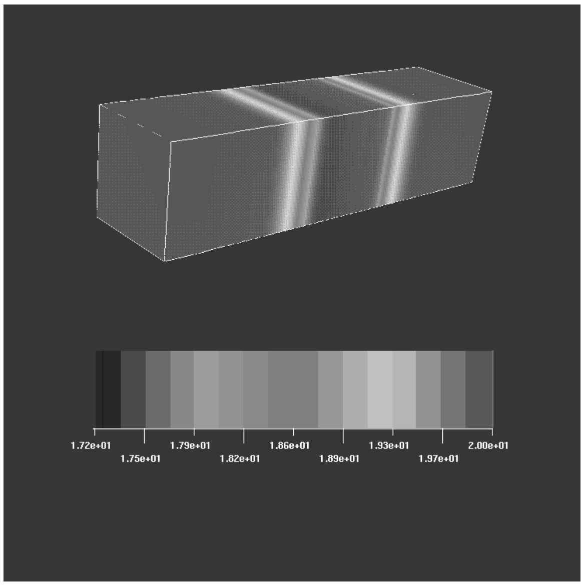

Figure 9: Monte Carlo estimate of electron concentration in the n^+nn^+ device structure. The legend is in terms of $\log_{10} n$.

simulation methods typically assume.

In addition to the n^+nn^+ structure, a three-dimensional n-MOSFET was simulated. This MOSFET device is $1.05\mu \times 1\mu \times 0.25\mu$ in dimension. The effective channel length is 0.25μ. The source and drain junction depths are 0.15μ, and they extend 0.67μ into the third dimension. For this example, $V_{gs} = 3.0V$ and $V_{ds} = 3.5V$, and the source and drain are doped $n = 1.0 \times 10^{20} cm^{-3}$. This MOSFET was simulated using 320,000 flight histories, with dynamic load balancing. On a 32 node CM-5, the total CPU time for simulation was 7162.1 seconds. On a 60 processor network of DEC 5000 workstations, the simulation required 5028.9 seconds. The result for electron distribution in the device is shown in Figure 12. Here, fringe effects can be seen which are due to the three-dimensional nature of the device. In particular, the inversion layer underneath the gate oxide can be seen to extend beyond the depth of the gate. This narrow-channel phenomenon causes inaccuracies in two-dimensional treatments of current flow.

6 Conclusions

Parallel and distributed architectures are shown to be effective for stationary Monte Carlo solutions of carrier transport in semiconductors. Although the initial spatially-based computations may be partitioned to avoid load imbalance, the execution times associated with the particle simulation phase are non-deterministic. This problem was address through static and dynamic load balancing algorithms. The relative effectiveness of these algorithms are analyzed through a randomized consideration of CPU loads. These loads, and correspondingly, the expected run-time of the simulator, are based on the distribution of CPU time required to simulate a flight history. This distribution is, in turn, dependent on the underlying structure and bias of the target device. The expected speedup of the dynamic load balancing algorithm over the static algorithm is shown to be related to the ratio between T_{fh} and $E[maxA_i]$, which is constant. For the simulation of the n^+nn^+ structure on the CM-5, the expected speedup was 22.32%. Overall, the network of workstations exhibited the best results, achieving scattering processing rates of 550,000 scatterings per second for the 3D

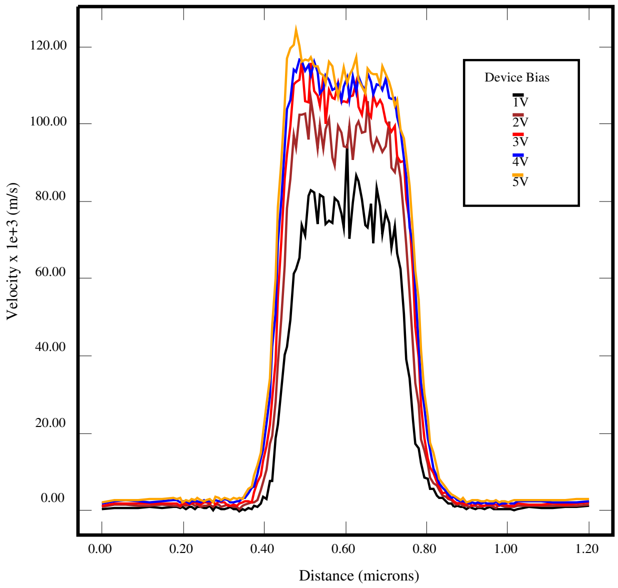

Figure 10: Average electron velocity in n^+nn^+ device, parameterized by applied bias.

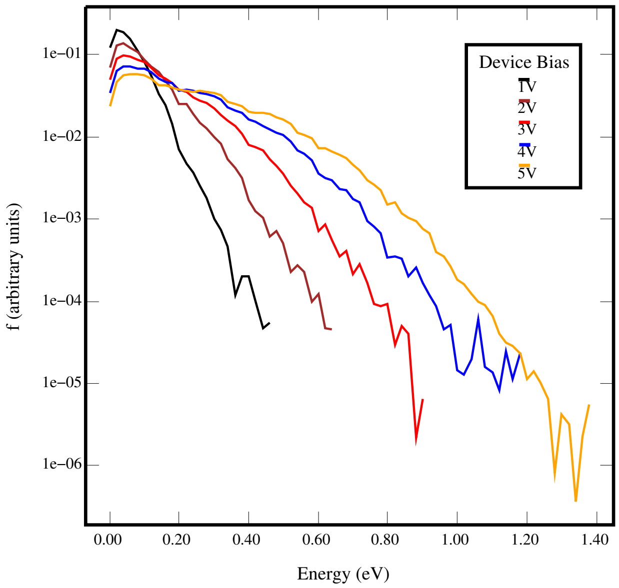

Figure 11: Electron energy distribution in n^+nn^+ device “channel”, parameterized by applied bias.

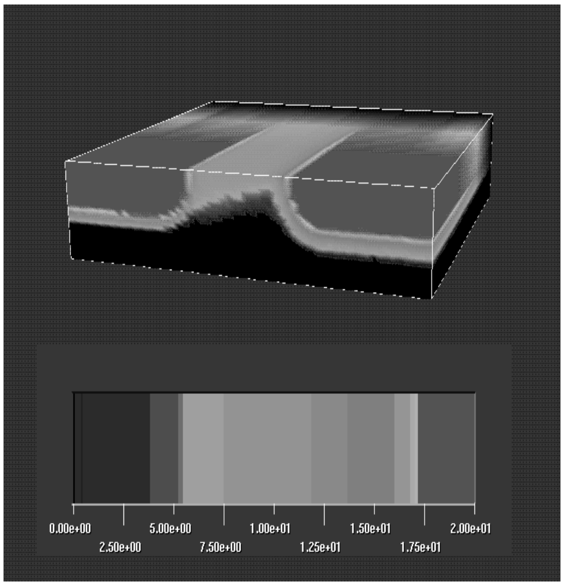

Figure 12: Monte Carlo estimate of electron concentration in a 3D MOSFET. The legend is in terms of $\log_{10} n$.

MOSFET. On the CM-5, rates of 364,000 scatterings per second were achieved. These compare favorably with vector performance [8](15,000-30,000 scatterings per second for typical MOSFET's on the Cray XMP) and the Connection Machine CM-2 [11](200,000 scatterings per second).

The scaling properties of this algorithm were investigated. Both the static and dynamic algorithms exhibited virtually linear scaling with problem size on a fixed architecture. The high cost of communications in workstation clusters generates congestion for the particle manager when the architecture is scaled with constant problem size per processor, since both task processing and outstanding particle requests are blocked during reply latencies. We are currently exploring statistical algorithms for flight history allocations of variable granularity. Communications may be reduced by initially appropriating large numbers of tasks per particle request and subsequently reducing this number as the simulation progresses, based on distribution-independent methods, such as Chebyshev inequalities. An alternative approach is to use empirical distributions of run-times during the course of simulation to control task granularity.

Currently, we are implementing methods for improving the variance of the Monte Carlo estimates, based on a variation of statistical importance sampling [5, 6]. The quality of the physical results improve, particularly in the estimation of phenomena based on distribution tails. The additional workload required for these algorithms is completely parallelizable, so that our parallel algorithm and the associated computational results are applicable in the presence of variance reduction as well.

7 Acknowledgements

This research was sponsored in part under the California MICRO program, Harris Corporation, Texas Instruments, Hewlett Packard, Motorola, Philips, Rockwell International, and Thinking Machines Corporation. Simulator development and experimental results were performed on the CM-5 machines at UC Berkeley, under National Science Foundation Infrastructure Grant CDA-8722788.

References

[1] T. Kurosawa. Monte Carlo Calculation of Hot-Electron Problems. In *Proceedings of the International Conference on the Physics of Semiconductors*, pages 424–426, 1966.

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

[3] J.M. Hammersley and D.C. Handscomb. *Monte Carlo Methods.* Chapman and Hall, 1983.

[4] A. Phillips and P. Price. Monte Carlo Calculations of Hot-Electron Energy Tails. *Appl. Phys. Lett.*, Vol. 30, pages 528–530, 1977.

[5] C. Jacoboni, P. Poli, and L. Rota. A New Monte Carlo Technique for the Solution of the Boltzmann Transport Equation. *Solid State Electronics*, Vol. 31, pages 523–526, 1988.

[6] L. Rota, C. Jacoboni, and P. Poli. Weighted Ensemble Monte Carlo. *Solid State Electronics*, Vol. 32, pages 1417–1421, 1989.

[7] M. Fischetti and S. Laux. Monte Carlo Simulation of Submicron Si MOSFET. In *Proceedings of SISDEP '88*, 1988.

[8] F. Venturi and et. al. A General Purpose Device Simulator Coupling Poisson and Monte Carlo Transport With Applications to Deep Submicron MOSFET's. *IEEE Transactions on CAD*, pages 360–369, April 1989.

[9] S. Sugino, C. Yao, and R. Dutton. Parallelization of Monte Carlo Analysis on Hypercube Multiprocessors and on a networked ews system. In *Proceedings of SISDEP '91*, pages 275–284, 1991.

[10] William R. Martin and Forrest B. Brown. Status of Vectorized Monte Carlo for Particle Transport Analysis. *The International Journal of Supercomputer Applications*, Vol. 1, pages 11–32, 1987.

[11] H. Sheng, R. Guerrieri, and A.L. Sangiovanni-Vincentelli. Massively Parallel Computation for Three-Dimensional Monte Carlo Device Simulation. In *Proceedings of SISDEP '91*, pages 285–290, 1991.

[12] H. Sheng, R. Guerrieri, and A.L. Sangiovanni-Vincentelli. A Generalized Self-Scattering Technique for Monte Carlo Simulation Suitable for SIMD Architectures. In *NASECODE X*, pages 24–25, 1994.

[13] H. Sheng, R. Guerrieri, and A.L. Sangiovanni-Vincentelli. A Generalized Self-Scattering Technique for Monte Carlo Simulation Suitable for SIMD Architectures. *COMPEL*, Vol. 13, pages 661–669, December 1994.

[14] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine.* The MIT Press, 1994.

[15] S. Selbeherr. *Analysis and Simulation of Semiconductor Devices.* Springer-Verlag, 1984.

[16] E. Sangiorgi, B. Ricco, and F. Venturi. MOS2: An Efficient Monte Carlo Simulator for MOS Devices. *IEEE Transactions on CAD*, pages 259–271, February 1988.

[17] D. Webber, E. Tomacruz, T. Toyabe, R. Guerrieri, and A. Sangiovanni-Vincentelli. A Massively Parallel Algorithm for Three-Dimensional Device Simulation. *IEEE Transactions on CAD*, September 1991.

[18] H. Rees. Calculation of Distribution Functions by Exploiting the Stability of the Steady State. *J. Phys. Chem. Solids*, Vol. 30, pages 643–655, 1969.

[19] R.W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles.* McGraw Hill, 1981.

[20] C. Moglestue. A Self-Consistent Monte Carlo Particle Model to Analyze Semiconductor Microcomponents of Any Geometry. *IEEE Transactions on CAD*, pages 326–345, April 1986.

[21] T. vonEicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. *Proc. of the 19th International Symposium on Computer Architecture*, May 1992.

[22] Thinking Machines Corporation. *CMMD Reference Manual*, December 1992.