THE 3-D MONTE CARLO SIMULATION OF A SEMICONDUCTOR DEVICE ON A HYPERCUBE MULTICOMPUTER *

U.RANAWAKE, C.HUSTER, P.LENDERS and S.GOODNICK
Department of Electrical and Computer Engineering
Oregon State University, Corvallis, OR 97331

Abstract

The efficient parallel implementation of a 3-D Monte Carlo device simulator is described. The parallel algorithm was implemented on a 64 node nCUBE multicomputer and its accuracy was validated by generating the static characteristics of a MESFET. Timing measurements were made to study the variation of the speedup of the parallel program as a function of the number of processors. We identify the sources of speedup loss and discuss several techniques for improving the speedup.

1. INTRODUCTION

The Monte Carlo technique is a numerical method for solving the Boltzmann's transport equation that is considered more physically accurate than device analysis tools based on the drift-diffusion (DD) model. However, models based on this technique are very computationally intensive, and therefore can greatly benefit from the vast computational power of today's parallel processors. In this paper we consider the parallel implementation of a 3-D device simulator on a hypercube multicomputer.

2. THE PARALLEL MONTE CARLO ALGORITHM

The flowchart of a typical Monte Carlo program for device simulation is shown in Figure 1 [1]. The parallel algorithm is an extension of the k-space Monte Carlo simulation with the addition of real space position of each simulated particle and the assignment of particle charge, using a cloud-in-cell scheme, to solve the Poisson's equation with the particle dynamics. The addition of the real space positions necessitates a geometric partitioning of the device, in which the grid is divided into three dimensional subgrids and assigned to processors using a gray code mapping. The parallel implementation of the Poisson's solution is based on an iterative method [2] that uses an odd/even ordering with Chebyshev acceleration. In order to make the communication during Monte Carlo simulation more efficient, each processor maintains a small buffer region of several layers of cells surrounding its subgrid. This region, called an *external interaction region* [3], is used to store the potentials of grid points owned by neighboring processors as well as to hold the particles that will move to these outer regions during the simulation of a time step. Communication between processors occurs during the solution of the Poisson's equation, the charge assignment, the contact simulation, the statistic gathering, and at the end of each time step when transferring particles

*Supported by NSF grant number ECS-8821107.

and fetching the potentials belonging to external interaction regions. The global communication that occurs during contact simulation and statistic gathering is handled efficiently using a binary tree routing scheme. The sequence of random numbers for each node process during Monte Carlo simulation is generated efficiently using the "Lehmer tree" concept [4].

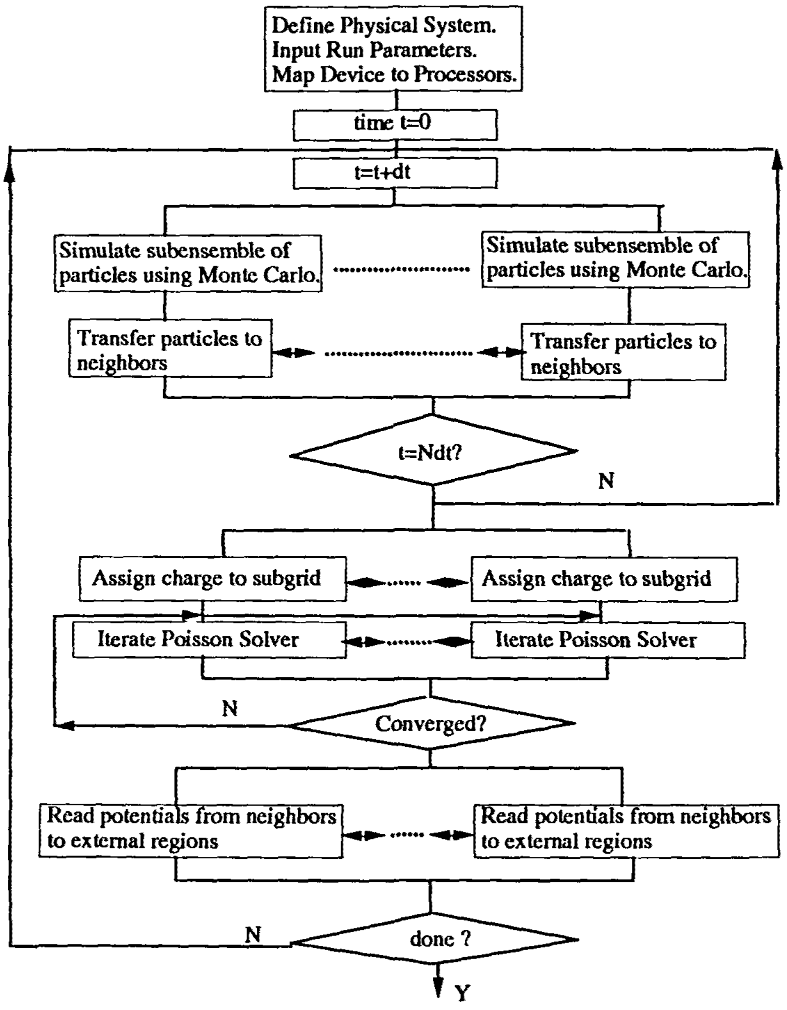
Define Physical System.
Input Run Parameters.
Map Device to Processors.
time t=0
t=t+dt
Simulate subensemble of particles using Monte Carlo.
Simulate subensemble of particles using Monte Carlo.
Transfer particles to neighbors
Transfer particles to neighbors
t=Ndt?
N
Assign charge to subgrid
Assign charge to subgrid
Iterate Poisson Solver
Iterate Poisson Solver
N
Converged?
Read potentials from neighbors to external regions
Read potentials from neighbors to external regions
N
done ?
Y

Figure 1: The Flowchart for Monte Carlo Device Simulation

3. RESULTS AND DISCUSSION

The parallel program was implemented on a 64 node nCUBE multicomputer at Oregon State University. The nCUBE consists of a host processor and an array of processing elements interconnected in a topology called a hypercube. Each node in the nCUBE system consists of a proprietary 32-bit VLSI CPU, 512 Kbytes of local memory, and 11 bidirectional Direct Memory Access (DMA) communication channels to handle communication with other nodes and the host processor. The nCUBE CPU performs arithmetic operations at the rate of about 100000 floating point operations per second [5].

As a prototype of the simulated device we have used a GaAS MESFET that is similar to the one described in [1]. The MESFET consists of a thin (0.16 μm) epitaxial layer of doped GaAs grown on a semi-insulating GaAs substrate. The epilayer and substrate doping are $10^{17} cm^{-3}$ and $5 \times 10^{15} cm^{-3}$ respectively. Three large pads of approximately $(200 \times 200)\mu m^2$ form the source, drain, and gate contacts. The separation between the source and the drain is about 3.8 μm. The source and the drain are assumed to be low resistance ohmic contacts, while the gate is assumed to form a Schottky barrier between the metal and the substrate. Only the active region of the device was modelled on the computer. The total device thickness was 5.12 μm while the thickness of the substrate was 0.16 μm. The test results were generated for a uniformly discretized grid of spacing 0.02 μm. To compare to 2-D results, only a few grid points in the z direction were considered.

The static characteristics of the simulated device are shown in Figure 2 These results were obtained for a $256 \times 16 \times 4$ grid using 24000 particles on a 64 node nCUBE. A time step of 0.05 ps was used for updating the potentials. The execution time to simulate 12 ps was about 5 hours. The distribution of electrons at the normal operating point $V_D = 3.0V$ and $V_G = -1.02V$ is shown in Figure 3. These results were generated for a $256 \times 16 \times 2$ grid using 12000 particles and 32 processors.

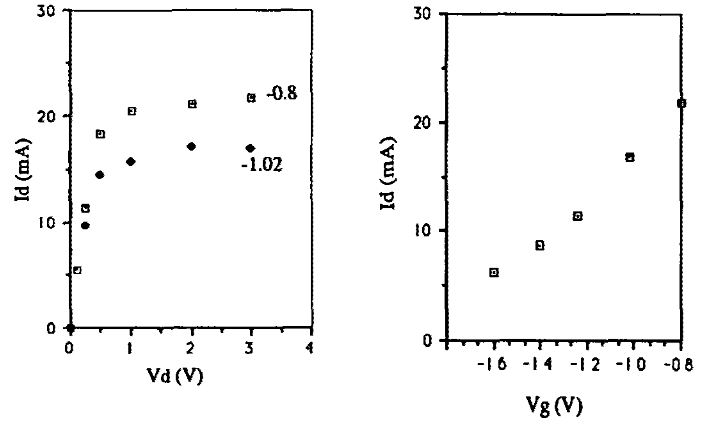

Figure 2: Static Characteristics

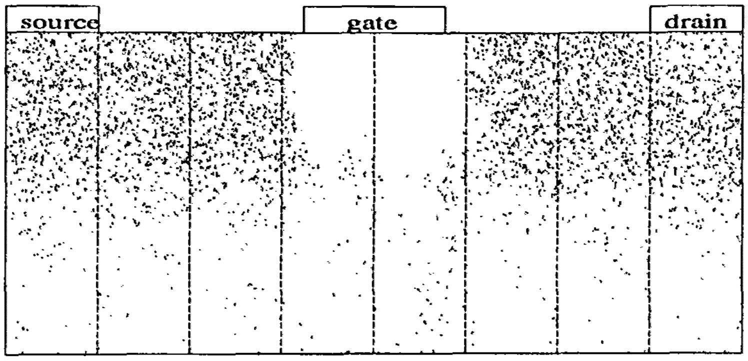

Figure 3: Distribution of Electrons

Timing measurements were made to study the variation of speedup as both the problem size and number of processors are varied. The speedup of a parallel program is defined as the ratio T_1/T_P, where T_1 is the execution time on a single processor and T_P is the execution time on P

processors. Figure 4 shows the speedups of the parallel program. For the test problem considered the speedup on all 64 processors is about 44. About 14 percent of this speedup loss is due to the communication overhead and the small serial component of the parallel program. The rest is due to the load imbalance on processors caused by the formation of the depletion region underneath the gate as shown in Figure 3. The execution times for processor 0 used in these computations are based on estimated values.

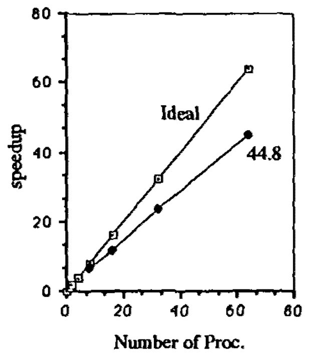

Figure 4: Speedups of the Parallel Program

Several methods are being investigated to increase the speedup. One is by dynamic balancing of load where the boundary surfaces shared by processors are adjusted by small amounts to reduce the load imbalance [3]. In this case the cost of rebalancing must be traded off against the lost time of having an imbalance in work load. The second method is based on the usual technique for enhancing rare events in Monte Carlo simulations, [6] where a particle entering a region with few particles is multiplied k number of times and a particle leaving that region is kept in the simulation with a probability $1/k$, with appropriate adjustment of the superparticle charge.

References

1. R. W. Hockney and C. Jesshope. *Parallel Computers - Architecture, Programming and Algorithms.* McgrawHill, New York, 1981.

2. R. W. Hockney and J. W. Eastwood. *Computer Simulation using Particles.* McgrawHill, New York, 1981.

3. S. B. Baden. *Run-Time Partitioning of Scientific Continum Calculations Running on Multiprocessors.* PhD thesis, University of California, Berkely, 1987.

4. W. R. Martin, T. C. Wan, T. S. Abdel-Rahmen, and T. N. Mudge. Monte carlo photon transport on shared memory and distributed memory parallel processors. *Journal of Supercomputer Applications*, 1(3):57–74, 1988.

5. J. L. Gustafson, G. R. Montry, and R. E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM Journal on Scientific and Statistical Computing*, 9(4), July 1988.

6. A. Phillips and P. J. Price. Monte carlo calculation of hot electron energy tails. *Applied Physics Letters*, 30:528–530, 1977.