Parallel Simulation of Electron-Solid Interactions for Electron Microscopy Modeling

S. J. Plimpton*, J. R. Michael#, A. D. Romig, Jr.#

Sandia National Laboratories
Albuquerque, NM 87185

* Parallel Computational Sciences Department, # Materials and Process Sciences Center

Key Words: parallel, Monte Carlo, electron, microscopy, random number generation

Abstract

A parallel implementation of a Monte Carlo algorithm for modeling the scattering of electrons in solids and the resulting X-ray production is described. Two important issues for accurate and fast parallel simulation are discussed - random number generation and load-balancing. Timing results for the parallel simulation are given which show even modest sized parallel machines can be competitive with conventional vector supercomputers for Monte Carlo trajectory simulations. Examples of parallel calculations performed to analyze specimen composition data and to characterize electron microscope performance are briefly highlighted.

Introduction

Analytical electron microscopy (AEM) is a tool for characterizing the spatial distribution of elements in multicomponent materials. One common method for performing such analysis is to detect the characteristic X-rays emitted by atoms in the microscope specimen (or sample) as they are excited by electrons from the microscope beam [Goldstein, et al. 1977]. With today's microscopes producing beam (spot) sizes in the 1 to 10 nanometer range and with detectors sensitive to single photon (X-ray) events, near atomic-level resolution of very low concentration (0.1 wt%) impurities can be achieved with AEM [Lyman 1986]. Thus it is a powerful technique for elucidating the microstructure of materials at surfaces, internal interfaces, and defects such as grain boundaries as well as for understanding diffusional growth and annealing processes in multilayer semiconductors, metals, and other complex materials.

The challenge facing the microscopist is to determine the actual composition and spatial distribution of second phases in a sample from the raw microscope data. Since even small spot size beams undergo considerable broadening due to scattering once they enter a material, the data taken at a point (or pixel) in the microscope image is actually composed of X-ray signals generated within a large spatial region of the sample. If the makeup of the sample is complex (and unknown), it can be impossible to deconvolve the data analytically and decipher the sample composition [Michael, et al. 1990]. A simpler, yet effective method for accomplishing this is by Monte Carlo simulation [Newbury and Myklebust 1979; Newbury and Myklebust 1981].

In this technique the sample composition and geometry as well as the microscope characteristics are input parameters to the model. The simulation computes the trajectories of electrons as they pass through the sample and the generated X-ray signal is the output. The signal can be compared to the experimental data and the input parameters adjusted if there is not good agreement. A sample can be characterized accurately by iterating this procedure.

In practice, the efficient use of such Monte Carlo calculations is severely limited by the large computation time needed to generate accurate statistics, particularly if the events of interest are rare (e.g., X-ray production from very small concentrations of impurities). For example, to model an electron beam scanning across an interface, dozens of beam impact points, each with 10^5 electrons might be simulated for one sample geometry and composition. If this is done repeatedly with different sample characteristics to deconvolve real data, more than 10^7 simulated electrons may be needed.

Because individual electron trajectories can be computed independently, AEM simulations are a natural candidate for parallel computing. Researchers have expended considerable effort to take advantage of parallel architectures on a variety of codes that fall under the general heading of Monte Carlo trajectory simulations [Martin 1988; Miura and Babb 1988; Frederickson, et al. 1987], often with very good results. Our initial interest in this work was sparked by knowledge of these efforts and from two additional factors. (1) We were users of a state-of-the-art serial AEM simulation code developed by researchers at the National Institute for Standards and Technology (NIST) [Newbury and Myklebust 1981] but were limited in the size of problems it could tackle. (2) We have access to parallel supercomputers (nCUBE and Intel machines) that seemed ideal for speeding up these kinds of computations. Using some of the physics routines from the NIST code as a starting point, we designed a new AEM simulation with parallelism in mind. In particular we made it capable of modeling very general geometries in anticipation of the broader range of complex samples that the speedier parallel machines would allow us to simulate. The resulting code can be run on a single processor (workstation or CRAY) or on any parallel machine that supports independent operation of its processors and a message-passing communications protocol - for example, the nCUBE 2, Intel Gamma and Delta, and the CM-5. The new code has proven to be efficient on all of these machines (2-3 times faster than the original NIST code even in the one-processor mode) except the CRAY, because it does not readily vectorize. However, it is so fast on the parallel machines that we have not thought it worth the effort to restructure the code for vectorization. More importantly, the power of the parallel machines has allowed us to model a variety of interesting effects that were previously intractable computationally. We have reported on some of the physics results elsewhere [Romig et al. 1990; Michael et al. 1991; Romig et al. 1991]; in this work we focus on the parallel issues related to implementing an AEM model, issues that are relevant to many kinds of Monte Carlo trajectory simulations, whether they be of charged particles, neutrons or photons.

In the following sections we first highlight the Monte Carlo algorithm appropriate for modeling electron scattering and X-ray production in solids. Then, its parallel implementation is detailed. Our choice of parallel architecture is justified in this section and a comparison of several parallel random number generators is made. A technique that proved effective for load-balancing the computations across many processors is also discussed. Finally, performance results for the simulation on several computers (parallel and serial) are given, and some example calculations performed with the parallel code are briefly described.

Monte Carlo Algorithm

The outline of a Monte Carlo trajectory algorithm is conceptually very simple. In our simulation, a 3-d geometry for a bulk or thin-film sample is constructed out of different vertical and/or horizontal layers, each with one or more elements in chosen concentrations. Each layer may also contain a spherical or block-shaped inclusion (of a different material) so that buried phases or voids can be simulated. Detector geometries and beam parameters (energy, spot size and shape, incident angle) are likewise specified. Individual electrons are then simulated as they impact the solid and traverse through sample. A possible trajectory is shown in Figure 1 for an electron entering a thin-film specimen consisting of a thin Cu layer in an Al matrix.

The primary (incident) electron's trajectory is built of small segments, each consisting of three parts: (1) a forward step during which the electron loses energy due to small-angle inelastic scattering, (2) X-ray production and resultant energy loss over the same step, and (3) an elastic (Rutherford) scattering interaction with an atomic nucleus that changes the direction of the electron, but not its energy. Optionally, a secondary electron may be created by a scattering event and it, in turn, may undergo further scattering events and produce X-rays. In principle an entire cascade of events could be produced by a single primary electron, but the energy regime of interest here (a few hundred kilovolts) is such that tertiary electrons have a negligible probability of retaining enough energy to produce X-rays, so they are safely ignored.

Each primary electron and its spawned secondaries are followed until they either backscatter (leave the specimen from the upper surface), are transmitted through the sample (in the case of a thin film), or lose enough energy that they can no longer excite X-rays. The code uses ray-tracing techniques to compute the intersections of electron paths with the various inclusions and layer boundaries. All of the physics modeled by the simulation is contained in the formulas used for energy loss of the electrons and the X-ray and scattering cross-sections, each of which is a function of electron energy and the material being interacted with. A fuller description of these is given elsewhere [Newbury and Myklebust 1981; Newbury 1986]. The Monte Carlo nature of the simulation is in the use of random numbers to calculate appropriately distributed step lengths, to choose between elastic scattering and secondary electron production events, and to select impact parameters for determining scattering angles.

X-rays that are emitted at various positions within the sample by the electron interactions are recorded and weighted by their probability of reaching the detector. When enough independent events have been simulated to tally good X-ray statistics, the computation is finished. The output of the simulation is the concentrations (wt%) of elements (derived from the X-ray statistics) that the microscope will detect with its beam positioned at a particular location on the specimen surface. Figure 1 illustrates the point made in the introductory section that even with the microscope beam positioned to the left of the Cu layer, electrons in the beam may scatter into the Cu and produce Cu X-rays. Thus the microscope “sees” a significant Cu signal even when the beam is far from the Cu layer. This is a typical characteristic of real microscopy data that a Monte Carlo simulation can be very helpful in analyzing, though large numbers of electrons may need to be simulated to generate a statistically significant number of interesting events.

Parallel Issues

The key idea that allows the Monte Carlo algorithm outlined in the preceding section to be implemented in parallel is that each electron can be simulated independently of all the others. A parallel computer with 1000 processors could compute 1000 electron trajectories simultaneously. Three important points should be noted, however: (A) The computation for each electron will, in general, follow a completely different logical path through the program and execute different physics routines. This is because the random nature of the Monte Carlo simulation causes any two trajectories to diverge in both energy and physical space. (B) The random numbers used to compute each of the trajectories must be statistically independent. (C) The CPU time required to simulate each trajectory may vary widely. One electron may backscatter immediately, while another undergoes numerous collisions and spawns many secondary electrons. Each of these three points is considered in turn.

(A) The first point implies that a multiple-instruction/multiple-data (MIMD) parallel computer (one that allows each processor to execute instructions independently of all the others) is a natural choice for the AEM simulation. On such a computer parallelism can be easily exploited by having each processor compute a subset of the electron trajectories. Because they run independently, each processor can execute its copy of the code in whatever sequence of operations is dictated by the electrons it is responsible for. By way of contrast, to extract parallelism (or vectorization) from a vector or single-instruction/single-data (SIMD) parallel machine one must group electrons together to perform one operation at a time on the entire group (“event-based” Monte Carlo, see [Martin 1988]). However in the AEM simulation discussed here (as in other Monte Carlo trajectory simulations) for complex sample geometries and high energy incident electrons there are enough kinds of low-probability events (causing branching in the code) to make an event-based strategy inefficient. In the SIMD case, for example, these events would require most processors to be idle while the special cases were computed. More significantly, this approach requires major changes in the algorithms and data structures of the serial program. Basic physics routines must be modified so that vectors of particles can be calculated on simultaneously. Notwithstanding these difficulties, there has been considerable effort by researchers [Miura 1987; Martin 1989] to vectorize serial Monte Carlo trajectory simulations for machines like the CRAY. Roughly tenfold increases in speed have been achieved using event-based techniques.

In our case, choosing a MIMD machine as the target architecture for the parallel AEM simulation meant the inner workings of the parallel code are no different that a standard serial implementation of a Monte Carlo trajectory simulation. In particular, the basic physics routines can be written as simple single-electron algorithms. The sequence of operations for the parallel code is as follows: (1) input the simulation parameters for a run of N total trajectories and send a copy of the parameters to all P nodes, (2) simulate N/P trajectories on each node, (3) combine the binned statistical data by sending messages between nodes, and (4) output the results. Step 2 requires all the computation time; step 3 can be performed very quickly on parallel machines like the nCUBE and Intel whose processors are interconnected in a hypercube or mesh topology.

(B) The second point - parallel random number generation - is one to which we paid particular attention. In the typical serial Monte Carlo algorithm an initial random number seed is supplied by the user. A sequence of random numbers is generated from the seed and used for successive trajectories; that is, the first electron uses the first few random numbers, the next trajectory uses a few more, etc. The drawback is that one does not know *a priori* what random number the n-th trajectory will start with. Hence a parallel simulation cannot duplicate the serial results.

To overcome this problem we implemented an idea discussed by Fox [Fox, et al. 1988] and similar to the random number "tree" concept detailed by Frederickson, et al [Frederickson, et al. 1987]. This method insures identical simulation results for any P, including P=1. The idea is as follows. Let 1024 be the largest number of nodes the simulation program would ever run on. When the simulation begins, the N electrons are divided up among the P processors by assigning each node 1024/P packets of N/1024 electrons each to simulate. For example, to simulate 10,000 electrons on 64 nodes, each node is assigned 16 packets of nine or ten electrons each. A unique random number seed is computed for each packet, and when that packet is simulated, its unique seed is used to initialize the random number generator. Note that there need only be one random number generator, a copy of which is owned by every processor. The packet seeds can be precalculated, with each node storing only the ones it will need for its packets. The key concept is that there is a one-to-one correspondence between seeds and packets. No matter what processor is assigned the packet, the same seed (and hence the same random numbers) will be used to simulate those electrons. Thus the simulation results will be identical whether all the electrons are run on a single processor (workstation or CRAY) or on 1024 processors of an nCUBE. The only requirements for this scheme to work are that (1) the 1024 unique seeds (one per packet, independent of P) can be computed from a single random number input by the user and that (2) the 1024 seeds initiate streams of random numbers that are independent of each other. We have implemented four different parallel random number generators (RNGs) using ideas from the literature that meet these two requirements and used the parallel AEM simulation as a test of their "goodness." We provide a few details of the four parallel RNGs and the testing procedure since they are useful techniques for many other parallel simulations that require random numbers.

Many RNGs are based on linear congruential generators of the form $x_{i+1} = (ax_i + c) \bmod m$, where $\{x_i\}$ is the random number stream and a,c,m are constants. In a review article, Park and Miller [Park and Miller 1988] give values of a,c,m which define a RNG with a period of 2^{31} that has been extensively tested and found to have excellent statistical properties. Its chief computational advantage is that the random numbers can be computed (in 64-bit arithmetic) with one or two lines of code without danger of overflow. We used it in our first parallel RNG and added a second, unrelated RNG to initially generate 1024 random numbers. These are used as the unique seeds for the Park and Miller RNG for each packet of electrons. This is a simple route to parallelism, but is somewhat dangerous since the second RNG could generate two packet seeds which are close together in the Park and Miller RNG sequence. If this happens the AEM simulation statistics created by those two packets of electrons will not be independent. As pointed out by Bowman and Robinson [Bowman and Robinson 1987], this correlation between sequences is often a potential problem with using "tree" methods for parallel RNG implementation.

A method proposed by Percus and Kalos [Percus and Kalos 1989] overcomes this problem. Their solution is to have one RNG and one initial seed, but use different c constants in the linear congruential formula to generate independent streams of random numbers. They prove "good" independence of a set of streams if the c_i are chosen to be small primes. For our second parallel RNG we computed the first 1024 prime numbers and assigned one to each packet of electrons as a c value. When a packet is simulated, the Park and Miller RNG is restarted with the user-input seed and the packet's assigned c value. The danger in this scheme is that every set of a,c,m values may not yield a "good" RNG in and of itself. Percus and Kalos showed that two streams with different c values will be independent, but there is no guarantee (without extensive testing) that each of the 1024 streams is as statistically "good" as the original Park and Miller RNG.

The third parallel RNG we tried was proposed by Bailey [Bailey 1991] and is based on a linear congruential generator with a different choice of a,c,m. It has a very long period of 2^{46} but is more expensive to compute since it protects against overflow by splitting each full precision operation into two half-precision parts. The benefit of the long period is that in a linear congruential generator it is a simple to compute the n-th random number in the stream. We used Bailey's RNG as our third parallel RNG in the AEM simulation by using the user-input seed to compute 1024 random numbers which were each separated by 2^{36} in the sequence. These became the unique starting seeds for the 1024 packets of electrons. Since each packet in our simulation needs far fewer than 2^{36} random numbers, this strategy insures each packet will use a random number stream that does not overlap with any other packet's stream. One detail is worth noting - to avoid the possibility of unwanted correlations in sequences separated by a power of two, we actually computed packet seeds separated by $2^{36} + n$, with n a small number, different for every packet.

The fourth parallel RNG we used was created by Marsaglia. We found the explanation and code listing in the review article by James [James 1990] to be most useful. This generator produces random numbers by combining two Fibonacci sequences with a subtractive technique. It is more expensive to compute than a simple linear congruential generator but has an extremely long period of 2^{144}. More importantly, any integer from one to 900,000,000 can be used to restart the generator and produce a stream of length 10^{30} that is “guaranteed” to be independent of any of the other 900,000,000 streams. To use this RNG in our parallel AEM simulation, we simply increment the user-input seed and restart the generator each time a packet of electrons is simulated.

The most important test of a RNG, parallel or otherwise, is how it performs in a particular simulation. This is because there is always the possibility some peculiar feature of the RNG will couple with a particular simulation to produce bad statistics. For our parallel AEM simulation we performed the following statistical test for each of the four parallel RNGs. Ten sets of N trajectories were run, each with a different user-input initial seed. The model problem was designed so that each run would produce a small X-ray signal generated from rare events (electrons reaching a distant interface). When N is small, there will be a significant variation in the observed signal from run to run. As N is made larger by a factor of M, the standard deviation ($\sigma = sqrt((\sum(x_i - \bar{x})^2)/10)$) in the ten signals x_i (where $\bar{x}$ is the mean of the ten values and the summation is from 1 to 10) should decrease as 1/sqrt(M) if the parallel RNG is producing random (uncorrelated) events. A smaller decrease in σ would indicate some kind of undesired correlation. We ran this test for each of the four parallel RNGs for N = 10^2, 10^4, 10^6 electrons. All the parallel RNGs “passed” the test since the computed σ decreased by roughly a factor of ten each time N increased by a factor of one hundred. We concluded that any of the four parallel RNGs are adequate for use in our AEM simulation.

(C) We now address the third parallel issue - load balancing - which can cause a parallel simulation to be inefficient (reduce the parallel speedup). It only turns out to be a problem in our AEM simulation when thin-film samples are modeled. Load imbalance can occur when most of the electrons run through the film quickly, but an occasional electron undergoes a high-angle scattering event and travels a long distance parallel to the film surface (as in Figure 1). Most of the processors then have to wait at the end of the computation for a few processors to calculate these anomalously long trajectories. In practice, parallel efficiencies as low as 13% were measured (on 1024 processors) due to this effect.

Our solution was to take advantage of the fact that for thin-film simulations, one typically wants to simulate many beam impact points on the surface (e.g., scan across an interface). The simulation was parallelized at a second level so that many different impact points are computed on simultaneously. Each processor thus computes trajectories for many more electrons before having to synchronize and communicate its results. This tends to average out the anomalously long trajectories so that most processors have a few of them to do. Parallel efficiencies rose to about 50% with this scheme. For typical simulations of thicker films and bulk samples, parallel efficiencies are about 80% to 90%, which includes the cost of I/O overhead, start-up computations that are duplicated on all the processors, and the inter-processor communication to combine the simulation statistics at the end of the computation. An alternative strategy to improve the load-balancing of the thin-film case would be to implement a master-slave scheme where very small packets of electrons are parcelled out to each slave processor by a master node and the slaves request more work when they finish their packets (see [Gustafson, et al. 1989], for example). Given the reasonable worst-case efficiency (50%) and the fact that a master-slave scheme would require additional communication overhead and necessitate changes in the parallel RNG methods outlined above, we did not implement such a scheme.

Performance Results

Timing results for the parallel AEM simulation on several machines are given in Table 1 for two benchmark problems. The parallel computer we use most for production runs of this code is an nCUBE 2 with 1024 processors. It is a MIMD machine with its nodes interconnected in a hypercube topology. Each node is a 1- to 2- megaflop CPU with 4 megabytes of memory (4 gigabytes total). The geometry and cross-section data for the AEM can be described very compactly so that each node can easily store the entire problem description including the statistical binning arrays in its 4 megabytes of memory. This is not true for all Monte Carlo trajectory simulations, and if one node can only store information for a limited spatial region, the parallel decomposition becomes much more complex.

The table compares the performance of the nCUBE with three machines: a CRAY Y-MP (single processor), a SPARC 2 workstation (with floating point hardware capable of about 5 megaflops), and a VAX 785, which is the machine the original NIST code was run on at our laboratory. The first benchmark problem is for 100,000 20-keV electrons into bulk Cu, a calculation that is done to compute a backscatter coefficient. The timing results are listed in the left columns of the table and the accompanying speedups are relative to the Y-MP time. The full 1024-node nCUBE is 118 times faster than a single Y-MP processor and about 2000 times faster than the VAX on this benchmark calculation. Comparing the CRAY and SPARC timings shows the relatively poor performance on the CRAY is because our AEM code does not vectorize at all (for the reasons discussed previously). However, it is worth noting that the same methods used to parallelize the nCUBE simulation could also be used to run the CRAY code efficiently on multiple Y-MP heads (as a MIMD parallel program). This would undoubtedly speed the CRAY timings up by nearly a factor of eight on a full Y-MP. The parallel efficiency of the 1024-node nCUBE simulation on this benchmark is 87%; that is, 1024 nodes were measured to be 890 times faster (.87x1024) than a single node. The one-node timing can be very accurately estimated with a shorter run since single-node timings scale very linearly with N for this problem. As discussed earlier the load balancing is good in this case because there is little variation in the time it takes to compute each long trajectory into the bulk sample. The lost 13% is mostly due to I/O, start-up computations, and the summing of statistics across the processors at the end of the simulation run.

The second benchmark calculation, shown in the righthand columns of Table 1, illustrates the poorer load balance for the thin-film case. For the benchmark, 100,000 100-keV electrons were simulated at each of 16 impact points across a Cr/Fe/Ni thin film interface (discussed in the next section). The nCUBE's speedup relative to the CRAY (31.6x) is not as great as in the bulk benchmark because the efficiency is lower (50% on 1024 processors) and also because the CRAY code is faster at performing some of the physics routines used in the thin-film simulation model. However, the parallel machines still offer a significant improvement over our previous capability.

Sample Calculations

In this section some examples of parallel AEM calculations are given. The first is a direct comparison with an experiment. A Cr-Ni-Fe alloy was annealed to produce α-phase precipitates in a γ-phase matrix. The concentration of Cr in the two phases is markedly different. The interface between the phases is known to be atomically discrete and very straight. Scanning a 1.0-nm electron beam across such an interface (with 89.1 wt% Cr on one side and 26.2 wt% Cr on the other) in a 150-nm thick film produces the concentration profile shown in the upper half of Figure 2 by the solid circles. A Monte Carlo simulation of the same geometry gives the diamond profile. where 10^5 electrons are simulated to generate each data point as the simulated beam is "scanned" across the interface. The agreement between the experimental and simulation data is excellent. In both cases the microscope images a much wider interface than physically exists because of beam broadening in the specimen. The lower half of Figure 2 shows four simulation curves for different film thicknesses. The equivalent experimental data is not shown but is given in [Michael, et al. 1990]. An interface "width" can be estimated for both the experimental and simulation data by calculating what portion of the profile is between the 10% to 90% bounds drawn on the figure. The 90% line is drawn nine-tenths of the way between the maximum and minimum Cr X-ray intensities, and similarly for the 10% line. The resulting "width" estimates are shown in the figure inset. The agreement is good, although most of the simulation results are slightly smaller. This is to be expected since the simulation is not affected by experimental difficulties such as orienting the interface exactly parallel to the electron beam, specimen drift, and errors in beam placement.

The second parallel Monte Carlo calculation characterizes the performance of two different microscopes, one with a field-emission source producing a very small spot size (1.0 nm) and one with a thermionic-emission source producing a larger spot size (20 nm). When these electron probes are scanned across a monolayer (0.3 nm) of Cu embedded in an Al thin film, the concentration profiles of Figure 3 result. Experimentally, such a sample might result from Cu segregated to a grain boundary in an Al metallization layer in an integrated circuit. The Cu X-ray signal is normalized to what it would be (100%) were the beams interacting with a pure Cu thin film. The smaller probe gives a much narrower, higher intensity peak for the Cu monolayer and thus would be able to resolve much smaller specimen features in practice. This is particularly interesting given the field-emission microscope's lower beam energy (100 keV vs. 200 keV). The spatial resolution is known to be a function of probe size, beam voltage, and specimen thickness. These results demonstrate that probe size is the dominant factor in determining the spatial resolution of the microscope.

Finally, our laboratory's electron microscopes are frequently used to study integrated circuit reliability problems. An example is where Al metallization lines have failed due to electromigration, stress voiding, or other mechanisms. The macroscopic cause of failure is often a void in the Al that grows to cause an "open" in the circuit. The electron microscope can be used to find and characterize these voids *in situ* without stripping off the overlaying SiO_2 insulator. The microscope images the buried void by detecting a small difference in the backscattered electron signal when the beam is positioned over the void vs. over the solid Al line. Physically, this difference occurs because the electrons passing through the void bury themselves deeper in the substrate and have less chance of backscattering. The Monte Carlo simulation can also detect this few percent difference in backscatter coefficient. We are using the parallel AEM simulation to answer several questions that are difficult to study experimentally: (1) What size of voids can be imaged and at what depth ? (2) What energy electron beam is optimal (maximizes the image contrast) for distinguishing a void from the surrounding Al ? (3) Does the thin layer of a heavier metal such as Ti or W that is often deposited under the Al to prevent Si spiking (diffusion into the Al) change the answers to (1) and (2) ?

Generating adequate statistics to answer these questions requires very large simulation runs of millions of electrons per impact point. These simulations are very time consuming because a bulk Si substrate must be included to calculate the backscatter coefficient correctly. Even on our 1024-processor nCUBE, it can take tens of minutes to run a single impact point. However, this compares to many hours on a CRAY. Thus the parallel machines are giving us the ability to examine this problem for the first time.

Conclusions

Parallel supercomputers provide a new level of computational power for Monte Carlo trajectory simulations such as the AEM model described here. Because such simulations break naturally into independent computational pieces requiring little inter-processor communication, they are ideal candidates for fast parallel performance on a MIMD parallel machines such as the nCUBE 2. Moreover, from an implementation standpoint, the great advantage of the MIMD model is that the basic physics algorithms can be expressed naturally as the interaction of one particle with its surroundings. This allows for great complexity in the geometry- and energy- dependent effects to be included without hurting the parallel performance as it often does for vector or SIMD machines.

The structure of many Monte Carlo transport simulations is similar to the one described in this paper. So long as the entire model description (geometry, cross section data, statistical binning arrays) will fit in the memory of each processor, similar parallel performance gains should be possible for neutron or high-energy electron-photon shower codes as well as codes simulating other low-energy electron-solid interactions, such as Auger electron production. Even on parallel machines smaller and cheaper than the 1024-node nCUBE 2, run times for such codes can be competitive with traditional supercomputers like the CRAY. In our case it has allowed us to examine a much larger set of interesting problems in reasonable turn around times.

Acknowledgments

The authors thank Bob Myklebust and Dale Newbury at NIST for giving us a copy of their serial Monte Carlo code and for help in the early stages of this work. This research was partially supported by the Applied Mathematical Science Program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia which is operated for the DOE under contract No. DE-AC04-76DP00789.

References

Bailey, D. 1991. The NAS Parallel Benchmarks. NASA Ames Technical Report.

Bowman, K., and Robinson, M. 1987. Studies of Random Number Generators for Parallel Processing. In Hypercube '87 Conference, Oak Ridge, p. 445.

Fox G., et.al. 1988. Solving Problems on Concurrent Processors, Prentice Hall (New Jersey), p. 210.

Frederickson, P., Hiromoto, R., and Larson, J. 1987. A Parallel Monte Carlo Transport Algorithm Using a Pseudo-Random Tree to Guarantee Reproducibility. Parallel Computing, 4, p. 281.

Goldstein, J.I., Costley, J.L., Lorimar, G.W., and Reed, S.F.B. 1977. Quantitative X-ray Analysis in the Electron Microscope. In Scanning Electron Microscopy, Vol. 1 (ed. O. Johari), IITRI (Chicago, 1977), p. 315.

Gustafson, J.L, Benner, R.E., Sears, M.P., and Sullivan, T.D. 1989. A Radar Simulation Program for a 1024-Processor Hypercube. In Supercomputing '89, IEEE, p. 96.

James, F. 1990. A Review of Pseudorandom Number Generators. Comp. Phys. Comm., 60, p. 329.

Lyman, C. E. 1986. Microanalysis Limits on the Use of Energy-Dispersive X-ray Spectroscopy in the Analytical Electron Microscope. In Microbeam Analysis - 1986 (ed. A. D. Romig, Jr. and W. F. Chambers), San Francisco Press, p. 434.

Martin, W.R. 1988. Particle Transport Monte Carlo on Shared-Memory and Distributed-Memory Parallel Processors. In Supercomputing '88, IEEE, p. 348.

Martin, W.R. 1989. Successful Vectorization - Reactor Physics Monte Carlo Code. Comp. Phys. Comm., 57, p. 68.

Michael, J.R., Williams, D.B., Klein, C.F., and Ayer, R. 1990. The Measurement and Calculation of the X-ray Spatial Resolution Obtained in the Analytical Electron Microscope. J. Microscopy, 160, p. 41.

Michael, J.R., Romig, Jr., A.D., and Goldstein, J.I. 1991. Thin Film X-Ray Quantitation: New Insights With the Aid of Monte Carlo Calculations. In proceeding of Electron Microscopy Society of America (EMSA) 1991 meeting, in press.

Miura, K. 1987. EGS4V: Vectorization of the Monte Carlo Cascade Shower Simulation Code EGS4. Comp. Phys. Comm., 45, p. 127.

Miura, K., and Babb II, R.G. 1988. Tradeoffs in Granularity and Parallelization for a Monte Carlo Shower Simulation Code. Parallel Computing, 8, p. 91.

Newbury, D.E. 1986. Electron Beam-Specimen Interactions in the Analytical Electron Microscope. In Principles of Analytical Electron Microscopy (ed. D. C. Joy, A. D. Romig, Jr., J. I. Goldstein), Plenum Press (New York), p. 1.

Newbury, D.E., and Myklebust, R.L. 1979. Monte Carlo Electron Trajectory Simulation of Beam Spreading in Thin Foil Targets. Ultramicroscopy, 3, p. 391.

Newbury, D.E., and Myklebust, R.L. 1981. A Monte Carlo Electron Trajectory Simulation for Analytical Electron Microscopy. In Analytical Electron Microscopy - 1981 (ed. R. H. Geiss), San Francisco Press, p. 91.

Park, S.K., and Miller, K.W. 1988. Random Number Generators: Good Ones are Hard to Find. Comm. of the ACM, 31, p. 1192.

Percus, O.E., and Kalos, M.H. 1989. Random Number Generators for MIMD Parallel Processors. J. of Parallel and Dist. Comp., 6. p. 477.

Romig, Jr., A.D., Plimpton, S. J., Michael, J. R., Myklebust, R.L., and Newbury, D.E. 1990. Application of Parallel Computing to the Monte Carlo Simulation of Electron Scattering in Solids: A Rapid Method for Profile Deconvolution. In Microbeam Analysis - 1990 (ed. J. R. Michael and P. Ingram), San Francisco Press, p. 275.

Romig, Jr., A.D., Michael, J.R., and Goldstein, J.I. 1991. X-Ray Spatial Resolution at Intermediate Voltages: An Assessment by Massively Parallel Monte Carlo Electron Trajectory Simulation. In Microbeam Analysis - 1991, in press.

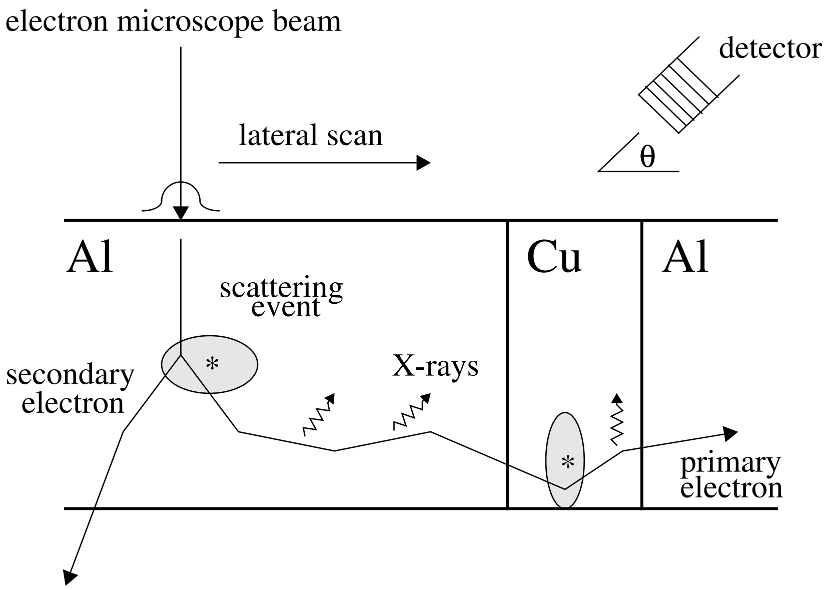
Figure and Table Captions

Figure 1: Diagram of the trajectory of an electron incident on an Al-Cu thin film with attendant production of secondary electrons and X-rays. Typically 10^5 electrons are followed for each impact point, and dozens of impact points are used to simulate a lateral scan across the Al-Cu interfaces.

Table 1: Timings (in seconds) for a bulk and thin-film benchmark problem on several machines. The CRAY timings are for one Y-MP processor; the SPARC 2 timings are for a SUN workstation; the nCUBE 2 timings are for 256, 512, and 1024 processors. The speedups for both benchmarks are relative to the Y-MP timings.

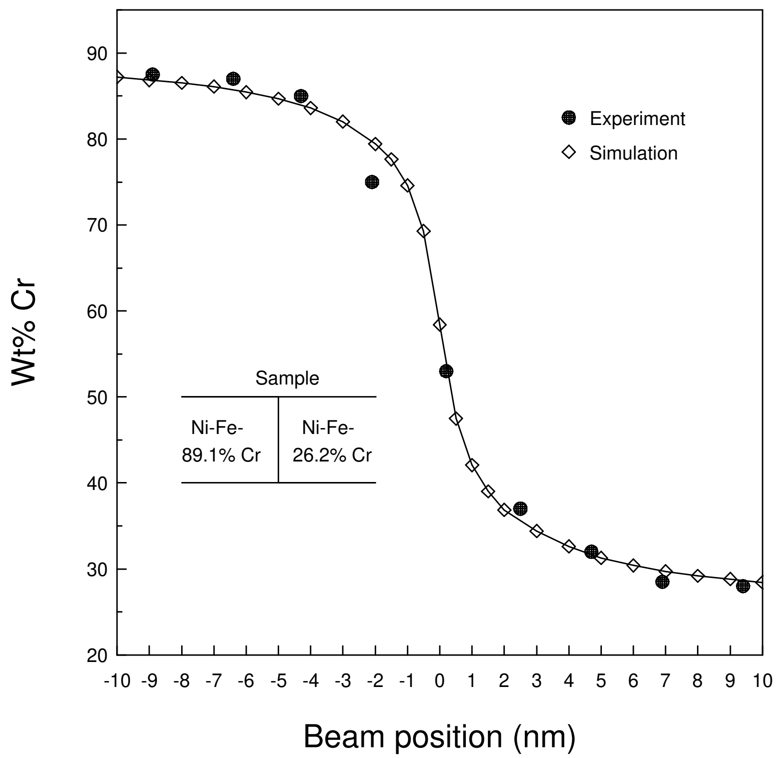
Figure 2: Upper: Detected Cr X-ray signal (normalized to concentration) as a 1.0-nm beam scans across an interface between two Cr phases. Both experimental and simulation data are shown. Lower: Simulation data for the same geometry for four different thin-film thicknesses. The observed width of the interface is a strong function of film thickness. 10^5 electrons were simulated to produce each data point.

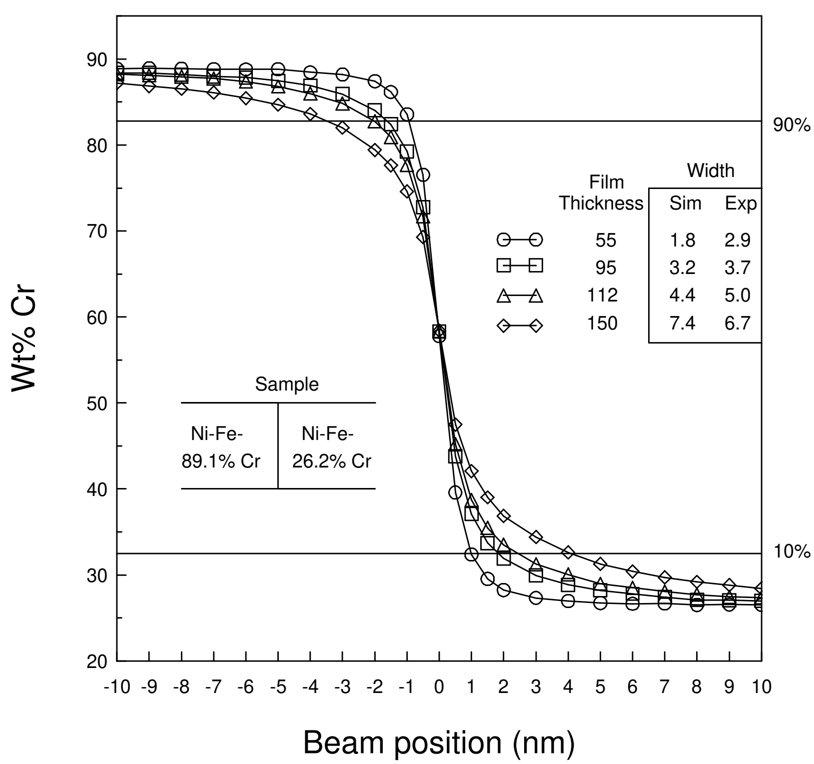
Figure 3: Simulated Cu X-ray signal as a function of beam position for scans across a thin Cu slice in an Al thin film (similar to Figure 1) for two different beam energies and spot sizes. The Cu signal is normalized to the 100% signal that would be detected for a pure Cu thin film. The much higher signal (62% vs. 4.9%) for the smaller spot size illustrates the importance of probe size in high-resolution AEM studies.

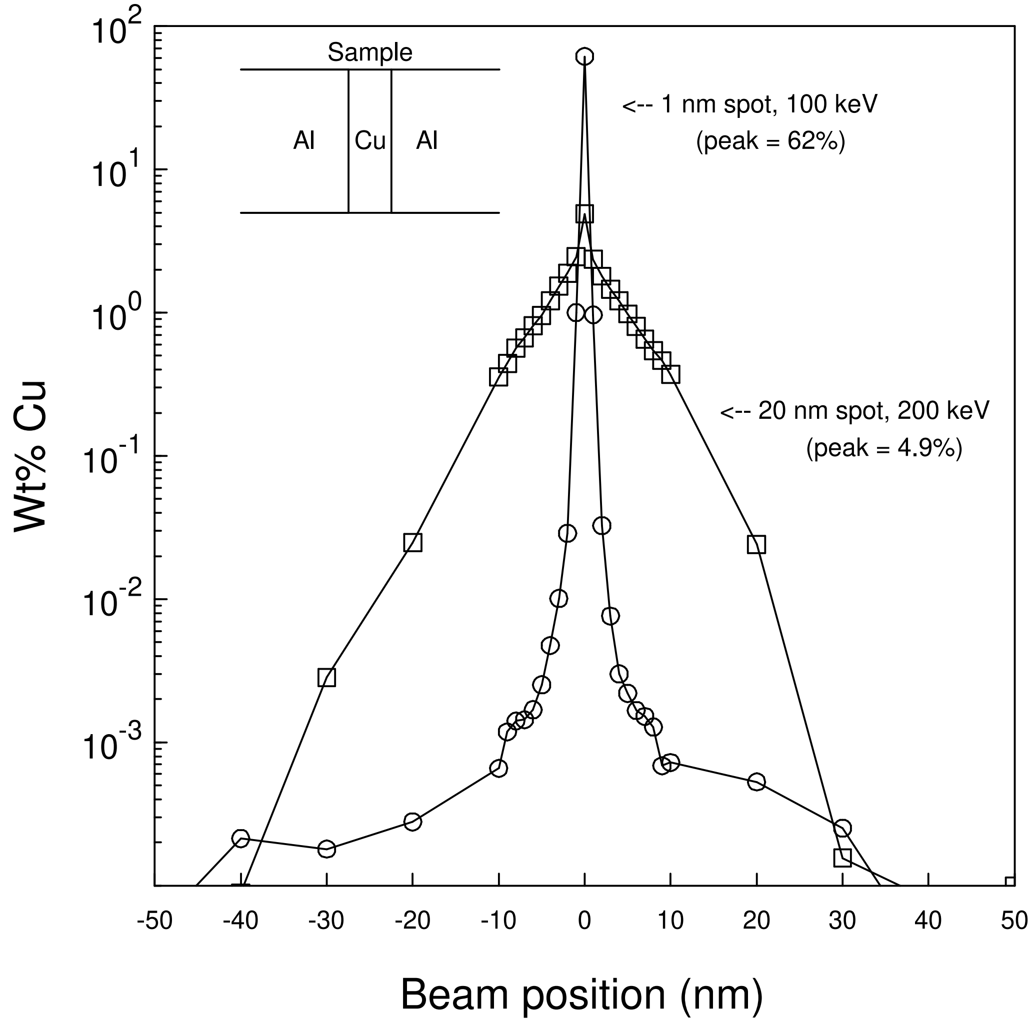


electron microscope beam
detector
lateral scan
θ
Al
Cu
Al
scattering event
secondary electron
X-rays
primary electron

Machine (Processors)	Bulk		Thin-Film	
	Time	Speedup	Time	Speedup
CRAY Y-MP (1)	3796	---	632.5	---
SPARC 2 (1)	10,293	0.37x	2787	0.23x
VAX 785 (1)	64,650	0.059x	21,600	0.029x
nCUBE 2 (256)	119.3	32x	50.6	12x
nCUBE 2 (512)	61.6	62x	33.1	19x
nCUBE 2 (1024)	32.1	118x	20.0	32x


90
80
70
60
50
40
30
20
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
Wt% Cr
Beam position (nm)
Experiment
Simulation
Sample
Ni-Fe-
89.1% Cr
Ni-Fe-
26.2% Cr


90
80
70
60
50
40
30
20
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
Wt% Cr
Beam position (nm)
90%
10%
Film Thickness
Width
Sim
Exp
55 1.8 2.9
95 3.2 3.7
112 4.4 5.0
150 7.4 6.7
Sample
Ni-Fe-
89.1% Cr
Ni-Fe-
26.2% Cr


Sample
Al
Cu
Al
<-- 1 nm spot, 100 keV
(peak = 62%)
<-- 20 nm spot, 200 keV
(peak = 4.9%)
Wt% Cu
Beam position (nm)
10^2
10^1
10^0
10^-1
10^-2
10^-3
-50
-40
-30
-20
-10
0
10
20
30
40
50