Parallelization of a Monte Carlo Ion Implantation Simulator

Andreas Hössinger, *Student Member, IEEE*, Erasmus Langer, *Member, IEEE*, and Siegfried Selberherr, *Fellow, IEEE*

***Abstract*—We present a parallelization method based on message passing interface (MPI) for a Monte Carlo program for two-dimensional and three-dimensional (3-D) simulation of ion implantations. We use a master–slave strategy where the master process synchronizes the slaves and performs the input–output operations, while the slaves perform the physical simulation. For this method the simulation domain is geometrically distributed among several CPU's which have to exchange only very little information during the simulation. Thereby, the communication overhead between the CPU's is kept so low that it has almost no influence on the performance gain even if a standard network of workstations is used instead of a massively parallel computer to perform the simulation. We have optimized the performance gain by identifying bottlenecks of this strategy when it is applied to arbitrary geometries consisting of various materials. This requires the application of different physical models within the simulation domain and makes it impossible to determine a reasonable domain distribution before starting the simulation. Due to a feedback between master and slaves by on-line performance measurements, we obtain an almost linear performance gain on a cluster of workstations with just slightly varying processor loads. Besides the increase in performance, the parallelization method also achieves a distribution of the required memory. This allows 3-D simulations on a cluster of workstations, where each single machines would not have enough memory to perform the simulation on its own.**

***Index Terms*—Distributed processors, ion implantation, Monte Carlo, parallelization, simulation.**

I. Introduction

ION IMPLANTATION is a very important, but concerning the simulation time also one of the most critical steps when simulating semiconductor device fabrication processes [1]. Due to the complicated structures and the small dimensions of modern semiconductor devices, Monte Carlo simulation methods often have to be used to describe nonplanarity effects, phenomena resulting from ion channeling and large tilt angles [2]–[4]. Moreover, they provide accurate point-defect distributions for rapid thermal annealing processes. To reach the expected accuracy, three-dimensional (3-D) simulations with sophisticated models [5], [6] have to be performed, especially for very shallow implantation conditions. By meeting all these requirements, the simulation times exceed one day or even more on high-end workstations, depending on the size of the structure and the complexity of the physical models.

For these reasons, the parallelization of the Monte Carlo ion implantation simulation is definitely desirable to avoid a bottleneck in the process simulation flow. For two-dimensional (2-D) applications the simulation time for a Monte Carlo ion implantation process step exceeds the simulation time for a diffusion or oxidation process step by a factor of more than ten. There is probably not such a big difference in the case of 3-D simulations, but at present it is difficult to state a serious comparison, because there is no optimized 3-D diffusion/oxidation process simulator available. First results [7] show that the ratio in simulation time is in the range of 1.5 to 3.

As an additional motivation, a cluster of workstations is usually available to perform process simulation and optimization. Therefore, we have concentrated on minimizing the communication overhead because it limits the performance gain especially if slow networks are used as it is the case for a cluster of workstations.

We present a parallelization method based on message passing interface (MPI) [8] which allows a distributed simulation on a cluster of single and multiprocessor workstations, which represents a common computer-aided design environment configuration. The merit of MPI is that it provides a comfortable and fast interface to define process communication and that it is available for a wide variety of hardware platforms. Our parallelization strategy keeps the communication overhead so low that it has almost no influence on the performance. All sophisticated physical models and methods [9], [10] developed for the single processor version can be used without modification.

The parallelization method has first been presented in [11] and evaluated for simple input structures. In this paper, we have also evaluated the parallelization strategy for the simulation of complex device structures with nonplanar surfaces consisting of various materials. Additionally, we have introduced a two-step concept for the distribution of the calculation tasks to optimize the performance for a loaded cluster of workstations and to optimally consider spatial variations of the trajectory calculation times due to spatially varying physical properties (different materials) of the target. As an extension of [11] the details of the parallelization strategy especially concerning the communication are presented and the parallelization overhead is analyzed.

II. Background

Our Monte Carlo ion implantation simulator MCIMPL is a multidimensional simulator which handles arbitrary shaped simulation domains consisting of various crystalline and amorphous materials. It is based on a binary collision algorithm

Manuscript received December 11, 1999; revised February 2, 2000. This work has been carried out within the SFB project AURORA, funded by the Austrian Science Fund (FWF). This paper was recommended by Associate Editor W. Schoenmaker.

The authors are with the Institut für Mikroelektronik, Technische Universität Wien A-1040 Vienna, Austria (e-mail: hoessinger@iue.tuwien.ac.at).

Publisher Item Identifier S 0278-0070(00)05300-8.

which means that the trajectories through the simulation domain of a lot of implanted ions are evaluated by successively calculating the interaction with atoms and electrons of the target material [12], [13].

The implanted particles sometimes remove target atoms from their original position and, thereby, generate recoiled particles which also move through the target. The simulation results which are stored in a histogram are the distributions of the implanted and of the recoiled particles (interstitials) and the distribution of the original position of the recoiled particles (vacancies).

Depending on the required accuracy and the target material, the recoil distributions (material damage) are derived either by an empirical model [14] from the distribution of the implanted ion or by a follow-each-recoil method [9]. Hereby, the trajectories of some or all recoiled particles are calculated rigorously as the ion trajectories. This means that additional moving particles are generated during the simulation. Worth mentioning is that the accumulation of the damage during the simulation which influences the particle trajectories, is considered. This is the biggest challenge for the parallelization strategy, because the simulation results influence the behavior of the simulation.

In order to improve the performance, MCIMPL uses two speedup methods. The trajectory-split method [10] for crystalline materials and the trajectory-reuse method [15] for amorphous materials. The trajectory-split method mainly increases the accuracy of the simulation by splitting a real particle into several virtual ones which partially share their trajectories. Thereby, several particles contributing to the particle distribution are generated from one real particle. This accelerates the overall simulation, even if the CPU time related to one implanted ion is significantly higher. In contrast, it is dramatically reduced by the trajectory-reuse method which stores complete ion trajectories and copies them to other parts of the simulation domain.

Considering all these properties of the simulator, it is obvious that the calculation time for trajectories varies significantly throughout the simulation domain, depending on the structure and the composition of the simulation domain.

III. Parallelization Strategy

We use a master–slave strategy based on MPI, where the master process provides all the input–output operations and controls and synchronizes the behavior of all slaves which perform the trajectory calculations. The most obvious parallelization strategy would be to successively distribute the trajectories among the available slaves. But this method has two significant drawbacks. On the one hand, the damage accumulation is not considered correctly. If the trajectories are arbitrarily distributed among the slaves, the order of trajectory calculation is nondeterministic which results in spatially varying damage accumulation. This effect is negligible as long as the number of slaves is significantly less then the number of ions. The number of ions simulated is in the range of some hundreds of thousands for two dimensional applications and some tenth of millions for three dimensional applications, while the number of workstations is typically of the order of ten. The more severe problem is that an arbitrary distribution of the trajectory calculation requires a continuous exchange of simulation results between the slaves, in order to correctly handle the influence of the damage on the ion trajectories. At least after each time step, the histograms where the simulation results are stored have to be updated. Even if very clever update methods are used, this requires a data transfer of several MB per time step. This transfer would only be acceptable for massively parallel computers but not for a cluster of workstations usually connected by 10 MBit/s or 100 MBit/s networks.

TABLE I
Typical Execution Times of the Single Processor Version on a DEC-600 Worksation for 3-D Simulations with Different Physical Models

Model	CPU time
Single atomic ion implantation with analytical damage models.	Some hours
Implantation of molecular ions with analytical damage models.	Several hours
Ion implantation simulation with the follow each recoil method.	One day

In our parallelization method, we explicitly introduce a transient simulation. The simulation time is divided into several time steps, and it is assumed that ions belonging to the same time step do not influence each other. This requirement is met if the distance between the entrance points is larger than the lateral range of the ions or if the number of ions per time step is small compared to the total number of simulated ions, which guarantees an almost constant damage within a time step. Therefore, the surface of the simulation domain is divided into square subdomains with a size slightly larger than the lateral range of the implanted ions. Depending on the total number of simulated ions, one to four ions are started from within each subdomain at each time step. The starting positions within the subdomains are equally distributed. For typical 3-D applications, the number of subdomains is larger than 1000; for 2-D applications, it is larger than 50.

Furthermore, the ions belonging to the same time step are not randomly distributed among the slaves to avoid the enormous communication overhead which frustrates the performance gain of the parallelization. Not only the trajectory calculation is distributed among several computers, but also the geometry of the simulation domain and the simulation results. Thereby, the communication between the slaves can be minimized as well as the memory requirement of the slave, because only a part of geometry description and of the simulation results has to be stored locally. Each slave is responsible for a prismatic scope whose base contains several of the subdomains mentioned above, as shown in Fig. 1. A slave only calculates parts of particle trajectories that are within its scope of responsibility. Communication only occurs if a particle leaves the scope of responsibility of a slave or if it moves in the vicinity of the border of the scope of responsibility.

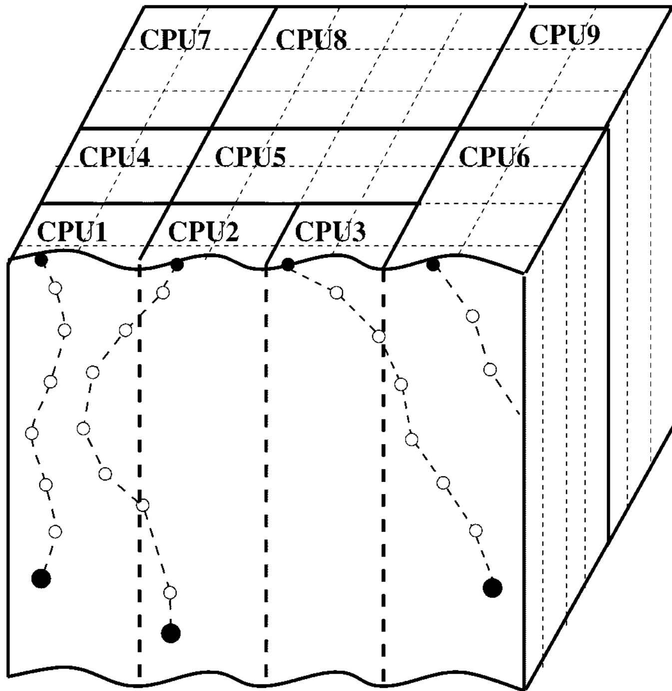

Fig. 1. Schematic presentation of the split of the simulation domain into subdomains and of the distribution of the subdomains among several processors. The small dashed lines lines denote the subdomains while the thick lines denote the scopes of responsibility of the slaves.

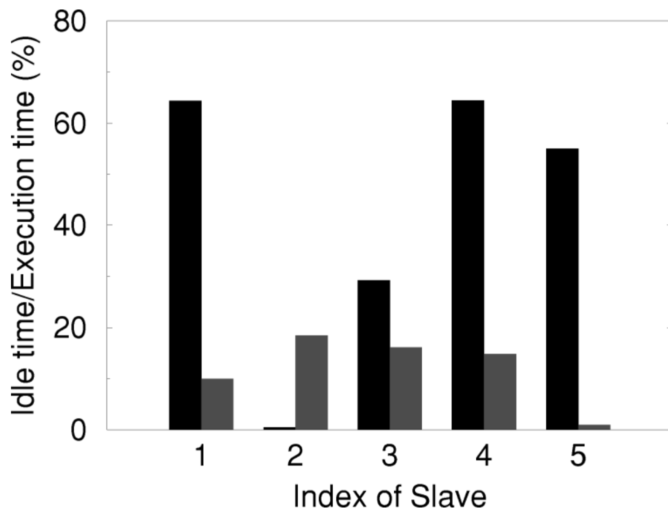

Fig. 2. Average idle times of the slaves for a simulation with five identical CPU's, using an arbitrary (dark blocks) and an optimized (light blocks) subdomain distribution.

IV. Distribution Scheme

The most critical aspect concerning the performance gain of our parallelization method is the distribution of the subdomains among the available processors. Due to the synchronization after each time step, the simulation time required for one time step is determined by the slowest slave, because all faster slaves are idle until the end of the time step (Fig. 2). In case of a poor distribution, the average idle times are of the order of the total simulation time and no performance gain can be achieved by the parallelization. A distribution scheme which guarantees similar simulation times for all slaves is obviously desirable. Therefore, the performance of a certain processor and its actual load have to be considered when the subdomains are determined. Nevertheless, it is not possible to reduce the average idle time to zero due to the statistical nature of the simulation problem.

Another problem is that due to spatially varying material properties which require the application of different models and algorithms the simulation time significantly depends on the entrance position of an ion into the simulation domain. Therefore, the actual simulation time of each slave can only be determined during the simulation, while the initial distribution can only be a more or less good guess.

To fully benefit from the parallelization strategy, the communication overhead has to be minimized by choosing a distribution with a minimum interface area between the slaves, because communication only occurs if an ion moves into the vicinity of an interface. The optimal situation would be if the scopes of responsibility of all slaves where prism with a square base, but this is generally not feasible.

In our parallelization method, we use a two-step concept when distributing the subdomains. For the initial distribution it is assumed that the trajectory calculation time is constant in all subdomains. Due to the fact the starting positions of the ions are equally distributed among the subdomains, the number of subdomains belonging to a certain slave i can be estimated by

$$N_i = \frac{N_{\text{sub}}}{\text{CPU}_i} \cdot \left(\sum_{j=1}^{N_{\text{CPU}}} \frac{1}{\text{CPU}_j} \right)^{-1}. \tag{1}$$

N_{CPU} is the number of available CPU's, N_{sub} is the total number of subdomains and CPU_i is the performance factor of slave i, which is proportional to the average time a slave needs to calculate one single ion trajectory. Starting with the fastest slave and continuing with slaves with decreasing speed the subdomains are distributed in the following way: First a subdomain k is selected with a maximum distance from already distributed subdomains l. This is done by assuming a repulsive potential between subdomains belonging to different slaves and selecting the subdomain k so that

$$\sum_{l=1}^{N_{\text{sub}}} \frac{\delta_{kl}}{|\vec{X}_k - \vec{X}_l|} \tag{2}$$

becomes a minimum. $\vec{X}_k$ and $\vec{X}_l$ are the coordinates of the centers of subdomains k and l. δ_{kl} is the Kronecker tensor.

The further subdomains are added to a slave by selecting the subdomain with the largest number of interfaces to the slave. If several subdomains meet this requirement, the subdomain with the lowest energy assuming a repulsive potential between subdomains belonging to different slaves and an attractive potential between subdomains belonging to the same slave is selected. Thereby, the average aspect ratio of the resulting scope of responsibility is kept low. Fig. 3 shows the resulting subdomain distribution on our alpha workstation cluster consisting of 19 CPU's (ALPHA 21164 with 333 MHz and ALPHA 21 064 with 200 MHz, 175 MHz, and 133 MHz), for an implantation into the structure shown in Fig. 4. The simulation domain with a size of 1 μm $\times$ 1 μm was split into 63×63 subdomains.

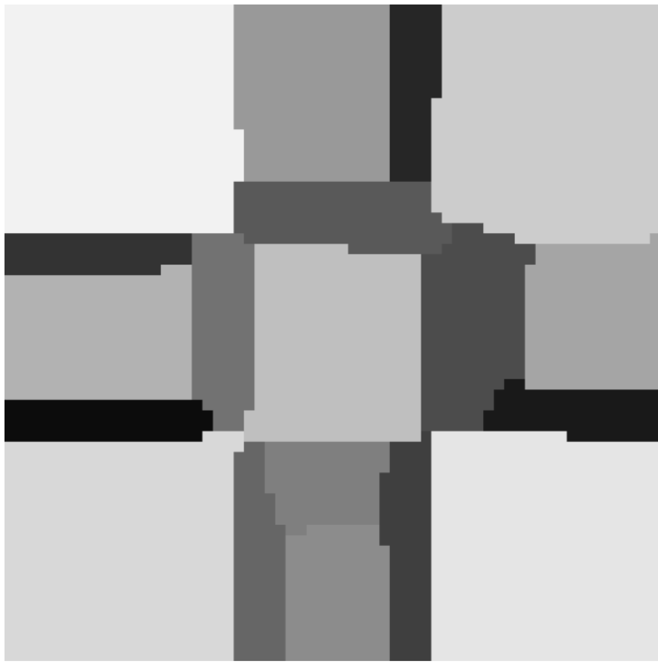

Fig. 3. Distribution of 63×63 subdomains on our alpha workstation cluster consisting of 19 CPU's. Each color denotes a CPU.

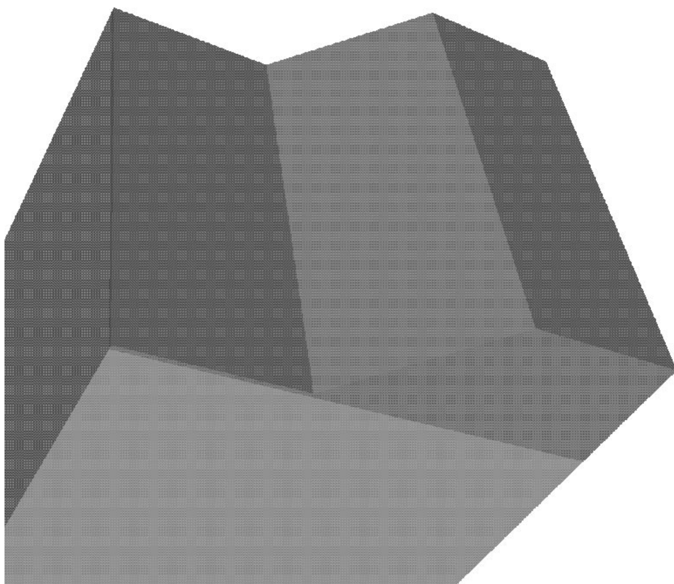

Fig. 4. Input structure for the ion implantation simulation with the subdomain distribution shown in Fig. 3. A block of silicon substrate partially covered by a thin scattering oxide layer and a thick oxide mask.

As will be explained in more detail in Section V, the current performance of each slave, considering the load and the variation of the physical properties, can be measured during the calculation of the first time step and using (1) the number of subdomains can be optimized for all slaves. In order to maximize the performance gain of the simulation the distribution is adapted based on the initial distribution. Only small modifications of the distribution scheme correctly consider spatial variations in the simulation time. In this way subdomains are successively moved from slaves with small idle times to slaves with large idle times until an optimized distribution scheme for the simulation is attained.

V. Simulation Flow

In order to demonstrate the behavior of the parallelization method, the complete simulation flow of the master process and of the slave processes is shown in the following.

A. Master Process

- Command line parsing.
- Reading of the input files.
- Initialization of the physical properties of the simulation domain, the physical models and the implantation conditions.
- Sending the initialization data and the geometry to the slaves: The amount of data transfer is mainly depending on the complexity of the input geometry and increases if damage information is reused from a previous implantation (0.1–200 MB).
- Creating the subdomains and evaluating the distribution scheme: Only the CPU performance of the slaves is considered for the distribution of the subdomains.
- Sending of the subdomains and the distribution scheme to all slaves (≈30 kB).

After the initialization, the master evaluates the quality (average idle time of the slaves) of the distribution scheme by initiating the simulation of one time step. The initial conditions for all ions are calculated and the properties of the ions are stored in stacks related to the slaves according to the entrance point into the simulation domain. Approximately just 0.2 kB are necessary to describe one single ion. The completed stacks are sent to the slaves before the master evaluates the momentary performance of the slaves, calculates an optimized distribution, and waits for the completion of the time step.

The performance of a slave and the end of a time step are determined by analyzing the responses of the slaves. Two types of messages are sent to the master during the simulation.

- Whenever a slave has finished all tasks a *Ready Message*, together with the number of processed (received) ions, is sent to the master.
- Before sending an ion to another slave, the master is informed about this activity by sending a *Ready Message* in common with -1.

While the master is waiting for the completion of the time step he collects all *Ready Messages* and decreases an internal counter with the number he received together with the *Ready Message*. At the beginning of the time step this counter is set to the number of ions that are sent to the slaves. If this counter is zero, the master knows that there is nothing left to do for the slaves. This slightly complicated protocol is necessary to correctly handle particles that are generated during the simulation either by the trajectory-split method or by the follow-each-recoil method, and to avoid errors due to communication delays. The master knows how many ions were sent to the network. The *Ready Message* in common with -1 informs the master that an additional particle is sent to the network. When all particles within the network are processed the time step is finished.

The performance CPU_i of slave i is derived from the first *Ready Message* the master receives from the slave, by measuring the time interval Δt between the sending of the ion

Fig. 5. Schematic description of the simulation flow of the master process and of a slave process. The thick arrows denote communication events between the master and the slave.

package of the first time step and the receiving of the *Ready Message*. This is a significant interval because no slave is idle during that interval

$$\mathrm{CPU}_i = \frac{\Delta t}{\text{Number of processed ions}}. \tag{3}$$

Nevertheless, it has to be mentioned that the measured CPU performance is influenced by the statistical nature of the simulation process. But the number of simulated ions per slave and time step is typically of the order of hundred and therefore, the measured performance is an average of a large number of statistical processes. Hence, the performance is not only valid for the first time step, but also for all further time steps.

Before the master starts the actual simulation he sends a *Reset Message* in common with the optimized distribution to the slaves to clear the simulation results of the initial time step, because the redistribution of the subdomains requires also the transfer of a lot of simulation results. This communication takes significantly more time than recalculating the first time step with the new distribution scheme. So, the master calculates the initial conditions of all ions of the first time step again, prepares them for distribution, and enters the following main control loop until the simulation is finished.

- Distributing the prepared ion packages among the slaves.
- Calculating the initial conditions for all ions of the next time step and preparing them for distribution.
- Waiting until all slaves have finished their calculations by collecting the *Ready Messages*.

The master terminates the main loop after the last time step is finished. He sends an *End Message* to all slaves to terminate their main simulation loop and collects all simulation results from the slaves (up to several hundred megabytes). To reduce collisions in the network due to the huge amount of data which are sent simultaneously from all slaves, the simulation results are collected piecewise. Finally, the master performs statistical analysis of the resulting doping and point-defect distributions, prepares the generation of the output, and writes the output files.

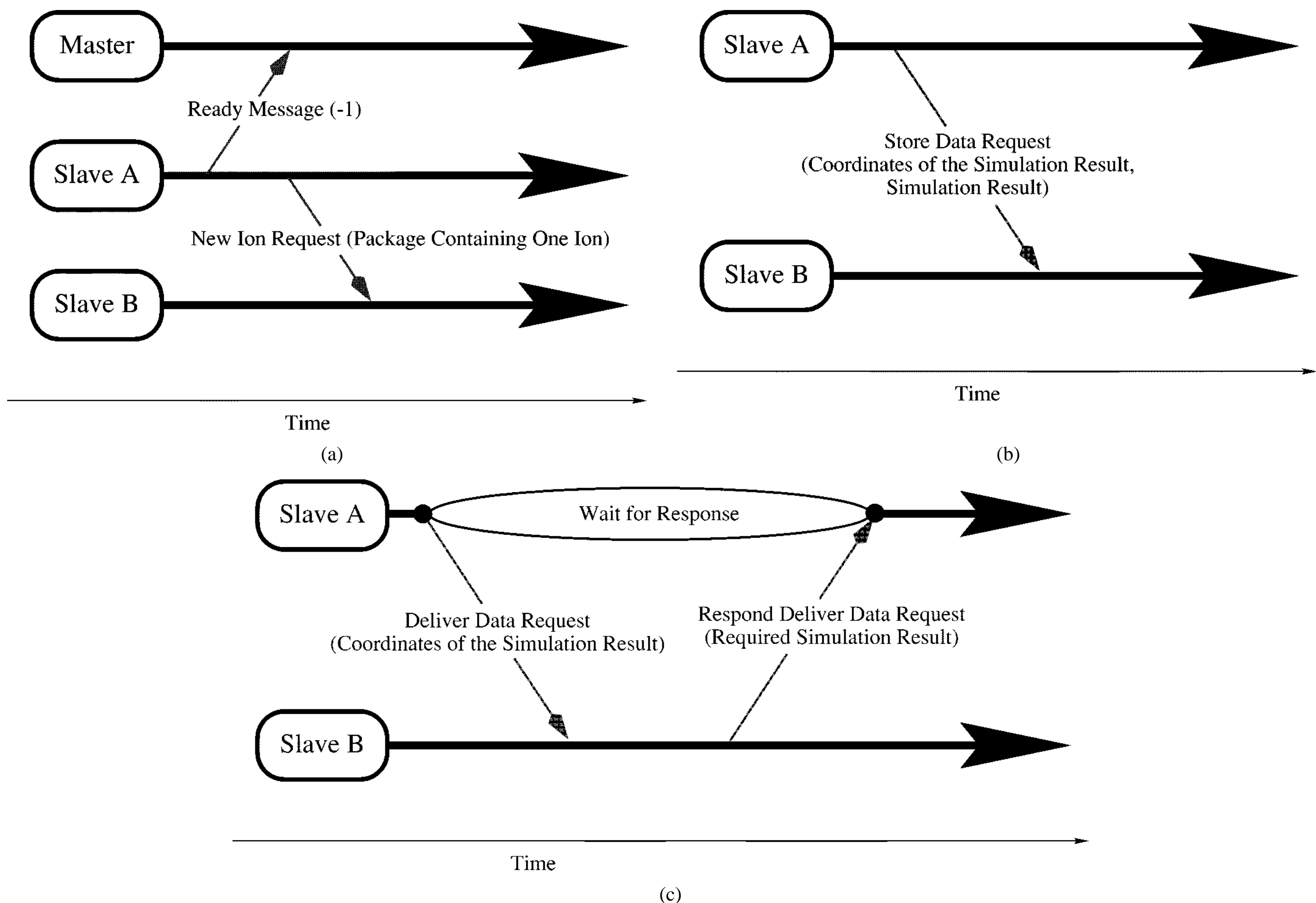

Fig. 6. Schematic presentation of the slave to slave communication events. (a) Transfer of an ion. (b) Storing simulation results outside the local memory. (c) Accessing simulation results from outside.

B. Slave Process

Similar to the master, the slave starts with an initialization process.

- Receiving the description of the simulation domain and of the implantation conditions for initialization.
- Receiving the initial distribution scheme. Thereby, each slave knows all scopes of responsibility, which allows a direct communication between the slaves.

Then, the slave immediately enters the main simulation loop, where his behavior is determined by requests he receives either from the master or from other slaves. Six different types of requests are handled.

- *Reset Memory*: The histogram where the simulation results are stored is cleared and a new distribution scheme is received. This request is used by the master to reset the simulation after he has evaluated the performance of the slaves.
- *Next Time Step*: The slave has to be informed about the beginning of a new time step because the trajectory stack used by the trajectory-reuse method has to be reinitialized after each time step.
- *Simulation Finished*: The slave leaves the main simulation loop, sends the simulation results to the master, and terminates operation.
- *Store Data*: The slave receives simulation results and the coordinates of where to store them and writes the data to the local histogram.
- *Deliver Data*: The slaves receives the coordinates of the required data and sends them to the slave who has asked for the data. This request is also processed during the calculation of an ion trajectory because the slave who has sent the request is blocked until he receives the response.
- *New Ion Package*: The slave can receive packages of several ions which are processed successively until they come to rest or leave the scope of responsibility of the slave. In that case, the master is informed by a *Ready Message* and the ion is sent to the neighboring slave. When all ions of a package are processed and no other request is pending, he sends a *Ready Message* together with the number of processed ions to the master, before he starts waiting for a new request. Besides the transfer of complete ions, two other types of communication events can occur. Simulation results located outside the scope of responsibility of the slave can be generated or required by certain models. For instance by the Kinchin–Pease Model [16] with a displacement in the vacancy and interstitial distribution [15]. Therefore, a method for nonlocal memory access is implemented. If simulation results have to be stored outside, a *Store Data Request* in common with the simulation data

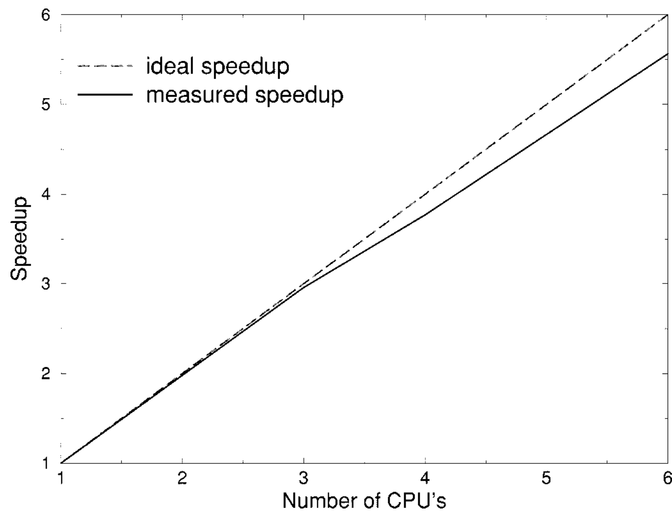

Fig. 7. Speedup as a function of the number of slaves compared to an ideal speedup.

and the coordinates of where to store them is sent to an appropriate slave. If simulation results outside have to be accessed, a *Deliver Data Request* is sent together with the coordinates of the required data. The slave has to wait for an answer, before continuing the simulation. Fig. 6 summarizes all possible slave to slave communications.

VI. Results

In order to estimate the performance gain of the parallelization, we have performed a 3-D simulation on a cluster of identical workstations using 1–6 slave processes. Fig. 7 shows the speedup as a function of the number of slaves. The speedup is determined by the ratio between the simulation time of the parallelized simulation and the single-processor version. In the case of just slightly varying processor loads, an almost linear performance gain is achieved and the overhead due to parallelization can be neglected.

However, although if the results are very promising, it will be necessary to implement a dynamic load-balancing strategy, because in normal operation conditions an almost constant load cannot be guaranteed on a cluster of workstations. The major problem of such a load-balancing strategy is that it requires the transfer of a huge amount of data between the slaves. Load balancing can be done by adding subdomains to faster slaves and removing them from slower slaves and, therefore, parts of the histogram which stores the simulation results have to be transfered. A strategy has to be developed which is capable of distinguishing between long-time and short-time load variations, because only the increase in the average idle times due to long time load variations warrants an efficient, dynamic redistribution of the subdomains.

VII. Conclusion

We have presented a parallelization method for a Monte Carlo ion implantation simulator which results in an almost linear performance gain on a cluster of workstations under just slightly varying load situations. The communication overhead can almost be neglected and, therefore, very fast networks are not necessary. Due to the fact that the memory requirement is distributed among several workstations, small workstations can be used for 3-D simulations.

References

[1] J. Lorenz, K. Tietzel, A. Burenkov, and H. Ryssel, "Three-dimensional simulation of ion implantation," in *Simulation of Semiconductor Processes and Devices*. Tokyo, Japan: Business Center for Academic Societies Japan, 1996, pp. 23–24.

[2] S. Tian, S. J. Morris, M. Morris, B. Obradovic, and A. F. Tasch, "Monte Carlo simulation of ion implantation damage process in silicon," in *Proc. Int. Electron Devices Meeting*, 1996, pp. 713–716.

[3] B. J. Obradovic, G. Balamurugan, G. Wang, Y. Chen, and A. F. Tasch, "Monte Carlo simulation of ion implantation into topographically complex structures," in *Proc. Int. Electron Devices Meeting*, 1998, pp. 513–516.

[4] M. Posselt, "3D modeling of ion implantation into crystalline silicon: Influence of damage accumulation on dopant profiles," *Nucl. Instrum. Meth. B*, vol. 96, pp. 163–167, 1995.

[5] G. Hobler and S. Selberherr, "Two-dimensional modeling of ion implantation induced point defects," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 174–180, Feb. 1988.

[6] S. Tian, M. F. Morris, S. J. Morris, B. Obradovic, G. Wang, G. Al F, G. Tasch, and Ch. M. Snell, "A detailed physical model for ion implant induced damage in silicon," *IEEE Trans. Electron Devices*, vol. 45, pp. 1226–1238, June 1998.

[7] V. Senez, S. Bozek, and B. Baccus, "3-dimensional simulation of thermal diffusion and oxidation processes," in *Proc. Int. Electron Devices Meeting*, 1996, pp. 705–708.

[8] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI—The Complete Reference: Volume 1, The MPI Core*. Cambridge, MA: MIT Press, 1998.

[9] A. Hössinger and S. Selberherr, "Accurate three-dimensional simulation of damage caused by ion implantation," in *Proc. 2nd Int. Conf. Modeling and Simulation of Microsystems*, San Juan, Puerto Rico, Apr. 1999, pp. 363–366.

[10] W. Bohmayr, A. Burenkov, J. Lorenz, H. Ryssel, and S. Selberherr, "Trajectory split method for Monte Carlo simulation of ion implantation," *IEEE Trans. Semiconduct. Manufact.*, vol. 8, pp. 402–407, Apr. 1995.

[11] A. Hössinger, M. Radi, B. Scholz, T. Fahringer, E. Langer, and S. Selberherr, "Parallelization of a Monte Carlo ion implantation simulator for three-dimensional crystalline structures," *Simulation Semiconduct. Processes Devices*, pp. 103–106, Sept. 1999.

[12] G. Hobler and S. Selberherr, "Monte Carlo simulation of ion implantation into two- and three-dimensional structures," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 450–459, May 1989.

[13] H. Stippel and S. Selberherr, "Monte Carlo simulation of ion implantation for three-dimensional structures using an octree," *IEICE Trans. Electron.*, vol. E77-C, no. 2, pp. 118–123, 1994.

[14] G. Hobler, A. Simionescu, L. Palmetshofer, C. Tian, and G. Stingeder, "Boron channeling implantations in silicon: Modeling of electronic stopping and damage accumulation," *J. Appl. Phys.*, vol. 77, no. 8, pp. 3697–3703, 1995.

[15] A. Hössinger, S. Selberherr, M. Kimura, I. Nomachi, and S. Kusanagi, "Three-dimensional Monte Carlo ion implantation simulation for molecular ions," *Electrochem. Soc. Proc.*, vol. 99–2, pp. 18–25, 1999.

[16] G. H. Kinchin and R. S. Pease, "The displacement of atoms in solids by radiation," *Rep. Progress Phys.*, vol. 18, pp. 1–51, 1955.

Andreas Hössinger was born in St. Pölten, Austria, in 1969. He studied physical engineering at the "Technische Universität Wien," Austria where he received the "Diplomingenieur"degree in January 1996. He is currently working toward the Ph.D. degree.

He joined the "Institut für Mikroelektronik,", Wien, Austria, in June 1996. In 1998, he was a Visiting Researcher at Sony/Atsugi. His research interests include process simulation with special emphasis on the simulation of ion implantation.

Erasmus Langer was born in Vienna, Austria, in 1951. He received the “Diplomingenieur” and doctoral degrees from the “Technische Universität Wien”, Vienna, Austria, in 1980 and 1986, respectively. In 1997, he received the “venia docendi” degree on “Microelectronics.”

He was with the “Institut für Allgemeine Elektrotechnik und Elektronik.” and in 1988 joined the “Institut für Mikroelektronik.” Since 1999, he has been Head of the “Institut für Mikroelektronik.” His current research topic is the simulation of micro-structures using high performance computing paradigms.

Siegfried Selberherr was born in Klosterneuburg, Austria, in 1955. He received the “Diplomingenieur” degree in electrical engineering and the doctoral degree in technical sciences from the “Technische Universität Wien” in 1978 and 1981, respectively.He received the “venia docendi” degree on “computer-aided design” since 1984.

Since 1988, he has been the Head of the “Institut für Mikroelektronik” and since 1999 he is Dean of the “Fakultät für Elektrotechnik.” His current topics of interest are modeling and simulation of problems for microelectronics engineering.