Proceedings of the IASTED International Conference
Applied Modelling and Simulation (AMS'99)
September 1-3, 1999, Cairns, Australia

A PARALLEL FINITE OCT-TREE FOR MULTI-THREADED INSERT, DELETE, AND SEARCH OPERATIONS

T. Binder and S. Selberherr
Institute for Microelectronics, TU Vienna
Gusshausstrasse 27–29, A-1040 Vienna, AUSTRIA
Phone +43-1-58801-36036, FAX +43-1-58801-36099
email: Thomas.Binder@iue.tuwien.ac.at

ABSTRACT

We present an object-oriented approach to distribute simulation data (like e.g., geometrical information, meshing information, distributed quantities) used for the simulation of semiconductor fabrication processes such as *Monte-Carlo* ion implantation [1], *etching* [2, 3], *diffusion* [4], and *oxidation* [5] on a heterogenous cluster of workstations. The approach is based on a finite *oct-tree* where the data are spread over the network. On SYMMETRIC MULTI PROCESSING (SMP) machines several elements can be handled in parallel (multi-threaded) to speed up insertion, searching, and deletion.

Keywords: parallel and distributed simulation, distributed data structures, microelectronics, modelling

1 INTRODUCTION

Three-dimensional simulations require large amounts of computer resources like CPU time and memory. In order to reduce resource consumption an often proposed solution is to parallelize the simulation task and utilize several computers instead of only one.

To allow for a distributed simulation not only the program must support parallelization, there is also the need for a distributed representation of data on the network. To illustrate this fact we take, for instance, a *Monte-Carlo* simulation of an ion implantation step in a modern Technology CAD (TCAD) environment like VISTA [6] or SIESTA [7]. The simulation domain is split into several sub-domains and an instance of the simulator is started on each host participating in the simulation. Each process holds only the data contained in its associated subdomain. Thereby the amount of memory required on each host is reduced. When the simulator partitions the simulation domain into sub-regions each resulting in roughly equal CPU time consumption there is also a reduction in the overall real time of the whole simulation.

2 STANDARD OCT-TREE

The parallel *oct-tree* is based on an object-oriented implementation of a standard (non-parallel) algorithm as suggested in [8].

The *oct-tree's* geometrical extension is defined by a cuboidal region called *root-leaf*. When elements are inserted this *leaf* is split recursively until a certain truncation condition (expressed by means of so called *terminal leafs*) occurs. The *terminal leafs* are not split any further. These *leafs* are used to (a) express the truncation conditions and to (b) hold the desired geometrical structure in memory. The *leafs* store references to the objects they represent. The following *leaf* types are in use:

- *point-leaf*
  Contains a reference to a point and a reference to the object(s) sharing this point.
- *line-leaf*
  Contains a reference to a line and a reference to the object(s) sharing this line.
- *face-leaf*
  References a face and the object(s) sharing this face.
- *solid-leaf*
  The *leaf* is completely surrounded by the object. Stores a reference to this object.

The *leaf* type used depends on the element to be inserted and is determined by applying certain geometry tests to the elements. The geometry tests were implemented with special emphasis on numerical stability. To cope with roundoff errors all tests were reduced to pure point comparisons. It is worth mentioning that, for a finite *oct-tree* the

accuracy of the arithmetic operations themselves is not as important as the uniqueness of the tests when they are applied for different sub-*leafs* of a *node*. Figure 1 illustrates this fact: The test, in which of the four drawn sub-*leafs* the element has to be inserted usually (except if one or more points of the line are completely within the *leaf*) results in a "line overlaps rectangle" test as depicted. The drawn "line-to-test" must either overlap the upper left or the lower right *leaf*, but must not overlap both (or none), or the correct *terminal leaf* cannot be determined and the insert method might end up with an endless recursion. Therefore, exact arithmetic as proposed in [9] would not guarantee the desired stability in our case. Since an epsilon based solution always results in a loss of resolution (maximum recursion depth) all tests are reduced to pure point tests instead, and half open intervals (Figure 2) are used for the *leafs*. Hence the maximum resolution only depends on the data type used to store a coordinate. We use an 8 byte double as defined in the IEEE standard 754 [10]. With a significant of 52 bits the maximum recursion depth is also limited to 52.

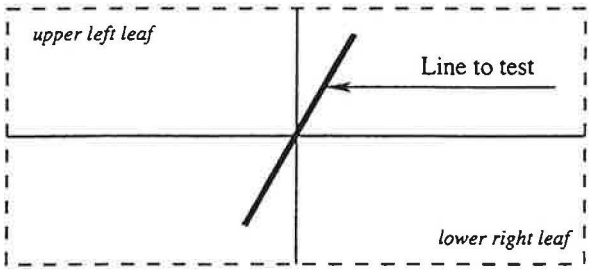


Figure 1: Projection of an *oct-tree leaf* onto a plane to illustrate the numerical problems occurring with geometry tests applied for different sub-*leafs*

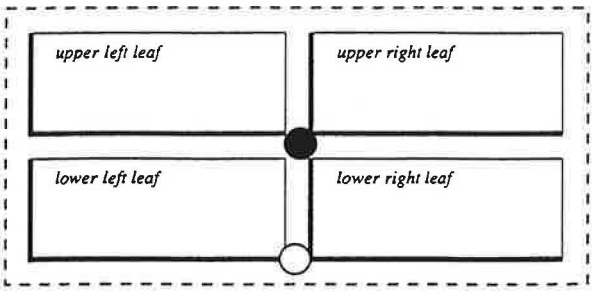


Figure 2: Half open intervals used in point comparisons. The dark drawn point is contributed to the *upper right leaf* whereas the grey shaded point is contained in the *lower right leaf*

3 PARALLEL *OCT-TREE*

The parallel *oct-tree* extends the capabilities of its non-parallel counterpart by the ability to transparently store *leafs* on several machines, thus allowing for the utilization of a whole workstation cluster. This functionality is achieved by extending the standard *oct-tree* by a new *leaf* type, the *network-leaf*.

The *network-leaf* is responsible for handling all network communications. The method used for the low-level communication between the hosts is hidden behind the *network-leafs'* interface and can be changed by sub-classing the interface and implementing a new *network-leaf* class. The current implementation uses the MESSAGE PASSING INTERFACE (MPI [11, 12, 13]) for the communication over the network.

The parallel *oct-tree* is implemented as a library which is linked against the application. Since MPI provides a powerful way to start a program on several machines no additional startup-code was necessary for the *oct-tree* itself. The communication is organized so that each host can contact any other host without the need for a master process.

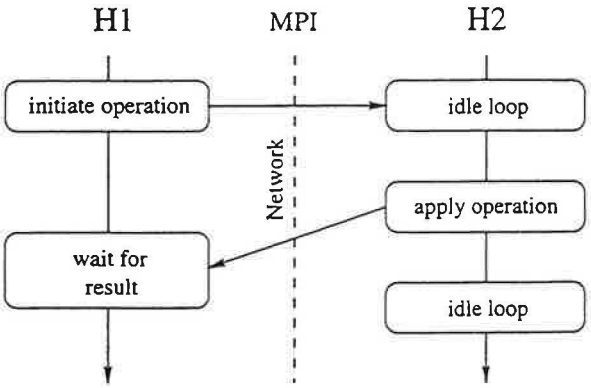


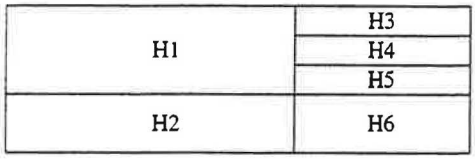
Figure 3: Protocol between two hosts

When a request for an operation (insert, delete, search) geometrically lies on a *network-leaf* (Figure 5(b)) the appropriate method of the *leaf* contacts the host associated with this region via a certain MPI message. The *oct-tree* instance on the remote host performs the required operation and returns the result via another MPI message type (Figure 3). Note, that every *oct-tree* instance has to periodically listen for incoming requests (idle loop). In order to avoid a deadlock in case two *oct-tree's* demand data from each other, the send and receive operations are non-blocking, that is, they return immediately.

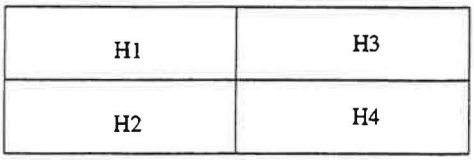
3.1 INITIAL PARTITIONING

When a new instance of a parallel *oct-tree* is created, the initial partitioning of the simulation domain needs to be specified. Figures 4(a) to 4(c) show possible configurations. There is no limitation on the number of regions/hosts which can be defined, however, the regions must be cuboidal and non-overlapping.

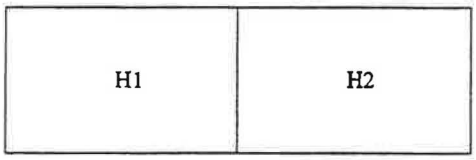
The partitioning should be chosen in such a way that each of the participating computers is equally loaded. Note, that it is left to the application to optimally balance the available computational resources. In order to allow the application to dynamically change the size of a region, i.e., to migrate parts of a region, statistics to detect an imbalance such as load or locality of point-locations are available.



(a) Configuration using six hosts



(b) Configuration using four hosts



(c) Configuration using two hosts

Figure 4: Possible configurations of parallel *oct-tree's*

3.2 INSERT/DELETE OPERATIONS

When an element is inserted into or deleted from the *oct-tree* a certain number of geometrical operations is computed. The number of operations depends on the type of *terminal leaf* and on the size and shape of the element. Since there are several instances of the *oct-tree* running on different machines elements which lie on disjunct geometrical regions can be inserted in parallel.

If two consecutive elements overlap the same *network-leaf* the insertion is done sequentially. It is quite obvious that, the order of the elements, directly influences the insertion performance. The situation is quite similar when elements are to be removed. Again, elements should be removed from distinct regions so that a maximum of the required geometrical operations is performed in parallel.

3.3 POINT LOCATION

Once all elements have been successfully stored, a transparent point-location (search) can be invoked on every host. Figures 5(a) and 5(b) show the two possibilities of a point-location from the host H1's point of view:

- Local point-location (Figure 5(a)) The element to be located is on the local host (H1).
- Remote point-location (Figure 5(b)). The element to be located is on a remote host (H2). The element is requested from the remote host.

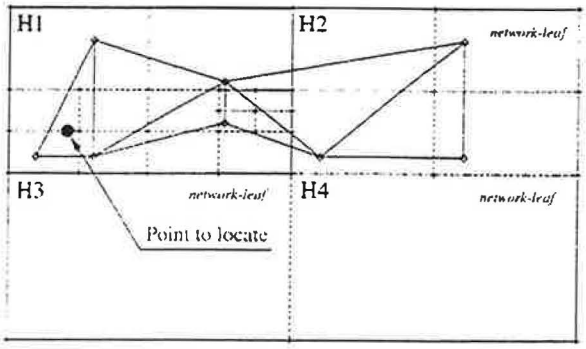
Note that the point-location is fully transparent, the application does not "know" on which host the data actually are stored. It is, however, worth mentioning that in order to keep the network traffic low, the application has to keep the operations as local as possible. Statistics about the locality of the operations are available so that the simulator can initiate a migration of a certain part of the simulation domain from one host to another. In case an element overlaps more than one host a copy of this element is kept on each computer, which conforms to a primitive caching algorithm.

3.4 MIGRATION

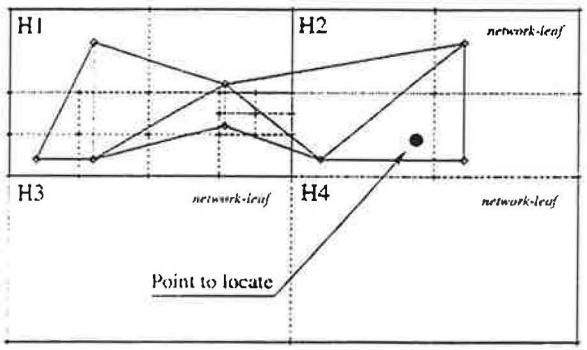
When the simulator discovers that the hosts are severely imbalanced it might be useful to change the size of a region on a certain host or even to migrate a whole region from one host to another. In this case the application simply requests a cuboidal region to be transferred. All elements are then deleted from the first region and inserted into the second one the same way a regular insert operation would take place. Note, that not only the elements have to be transferred over the network, but also the geometrical tests are performed twice (for the delete and for the insert operation). This rather drastic measure makes sense in long lasting simulations where the load shifts from one host to another during the simulation and, as a consequence, the communication overhead increases dramatically.

4 CONCLUSION

We present a new method to distribute simulation data as they occur in standard semiconductor fabrication pro-



(a) local point-location



(b) remote point-location

Figure 5: Projection of an *oct-tree leaf* onto a plane. The grey shaded part denotes the host (H1) where the point-locations take place

cess simulations like *Monte-Carlo* ion implantation, *etching*, *diffusion* or *oxidation* over a cluster of heterogenous workstations. The method assists the simulators in performing parallel simulations.

The parallel *oct-tree* is used in our parallel wafer state server which integrates several services like gridding, different file formats, visualization. This wafer state server is used to assist the TCAD frameworks in simulating whole process flows.

The chosen programming language for implementing the parallel *oct-tree* is *C++*. This language facilitates a full object-oriented design as employed in the *oct-tree's* core classes as well as a good level of abstraction from operating system specifics like multi-threading or network communication. Due to the very object-oriented design of our standard *oct-tree* the parallel extensions were implemented without the need to change or even recompile the standard *oct-tree*.

References

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

[2] W. Pyka, R. Martins, and S. Selberherr, "Efficient algorithms for three-dimensional etching and deposition simulation," In Meyer and Biesemans [14], pp. 16–19.

[3] R. Mlekus, Ch. Ledl, E. Strasser, and S. Selberherr, "Polygonal geometry reconstruction after cellular etching or deposition simulation," in *Simulation of Semiconductor Devices and Processes*, H. Ryssel and P. Pichler, Eds., Wien, 1995, vol. 6, pp. 50–53, Springer.

[4] M. Radi, E. Leitner, E. Hollensteiner, and S. Selberherr, "Amigos: Analytical model interface & general object-oriented solver," in *Basics and Technology of Electronic Devices*, K. Riedling, Ed., Grossarl, Austria, Mar. 1997, Gesellschaft für Mikroelektronik, pp. 57–60, Proc. of the Seminar "Grundlagen und Technologie Elektronischer Bauelemente".

[5] M. Radi, *Three-Dimensional Simulation of Thermal Oxidation*, Dissertation, Technische Universität Wien, 1998.

[6] R. Strasser, Ch. Pichler, and S. Selberherr, "VISTA - a framework for technology CAD purposes," in *9th European Simulation Symposium*, W. Hahn and A. Lehmann, Eds., Passau, Germany, Oct. 1997, pp. 450–454, Society for Computer Simulation International.

[7] R. Strasser and S. Selberherr, "Parallel and distributed TCAD simulations using dynamic load balancing," In Meyer and Biesemans [14], pp. 89–92.

[8] M. Mäntylä, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, 1988.

[9] Jonathan Richard Shewchuk, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," *Discrete & Computational Geometry*, vol. 18, no. 3, pp. 305–363, Oct. 1997.

[10] IEEE Std. 754, "IEEE Standard for Binary Floating Point Arithmetic," *Reaffirmed 1990*, 1985.

[11] Message Passing Interface Forum, "MPI: A message-passing interface standard," *International Journal of Supercomputer Applications, 8(3/4)*, 1994.

[12] Message Passing Interface Forum, "MPI: A message-passing interface standard," *Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville,TN*, 1994.

[13] William Gropp, Ewing Lusk, and Anthony Skjellum, "Using MPI: Portable Parallel Programming with the Message Passing Interface," *MIT Press*, 1994.

[14] K. De Meyer and S. Biesemans, Eds., *Simulation of Semiconductor Processes and Devices*, Leuven, Belgium, 1998. Springer.