P-29

# High Performance Semiconductor Device Simulation on Shared Memory Parallel Computers

Mounir Hahad and Peter Hopper
Silvaco International
4701, Patrick Henry Drive, Santa Clara, CA-95054
e-mail: {mounirh,peteh}@silvaco.com, phone: (+1) 408.654.4326, fax: (+1) 408.496.6080

## 1 Introduction

Recently a large number of practical, low cost, parallel computers have been made available on the market from all of the major workstation vendors. Although these supercomputers (called shared memory symmetric multiprocessors) provide a single memory shared by all the processors, they still require specific code development in order to run a single job in parallel. Fortunately, shared memory programming, as opposed to distributed memory programming, is very close to the usual sequential programming style. While shared memory programming engineers are focusing on getting the most of the computing power, distributed memory environments are still struggling toward a programming language standardization. Consequently, the delay to the availability of codes that would make use of such shared memory architectures is more likely to be minor as compared to distributed memory platforms.

Device simulation is a time consuming, computation intensive class of applications. Thus, it is perfectly suited to be adapted to take full advantage of shared memory multiprocessor computers. Although Silvaco's Virtual Wafer Fab (VWF) Automation Tools could make full use of this computer architecture by task farming statistical simulations onto a parallel machine, sending a job to each processor for example, the problem of making single interactive jobs run faster, still remained.

Silvaco has recently developed a new parallel version of *ATLAS* (Silvaco's device simulation tool) now capable of running much faster on these types of machines. The scheme of parallelization employed by Silvaco has focused on the general case of 2D Device simulation. All forms of 2D device simulations benefit from this approach. The following article describes some of the types of general structures that can be simulated in parallel.

Bench marking has been completed on machines ranging from single processor Silicon Graphics (SGI) machines, to small desk-side server machines to large scale supercomputers. All tests have been performed at the super computer facilities within SGI.

Today, Parallel *ATLAS* is available on SGI machines, offering the very high performance as indicated below. Further current work is going ahead to make available the same parallel code on all other mainstream parallel machines shortly. These machines will include those standard parallel machines from Convex, HP and SUN.

## 2 Problem Description

At the core of *ATLAS* exist a set of physical models the simulator relies upon to predict all sorts of semiconductor device behavior. The equations describing the underlying physics (there can be up to 6 coupled equations to solve: Poisson, electron and hole continuity, electron, hole and lattice temperature.) are solved using a finite difference method. These methods discretize the partial differential equations (PDEs) to reach an approximate solution to the problem. The finite difference method implemented within *ATLAS* is by far the most time consuming phase during any simulation run and thus, has been the natural target for the parallelization work.

The first step of a finite difference approach is to overlay a mesh on a device structure geometry for discretization purposes. Within Silvaco's Virtual Wafer Fab (VWF) integrated environment, the meshing can either be generated in *ATHENA* during the process simulation, by *DevEdit* (a device meshing tool) or in *ATLAS* itself. As we will discuss later, the speed of the parallel simulation may vary depending on the origin of the mesh due to different meshing algorithms.

Once the meshing has been performed, the simulator uses an iterative method to solve a non-linear system of equations. Each step of the non-linear method (Newton, Gummel or a combination of both) requires the following two steps:

- evaluation of the sparse Jacobian matrix, which actual size depends on the number of nodes in the mesh and the number of PDEs being solved. We will refer to this operation as the assembly phase.
- solution of a large sparse linear system of equations, which *ATLAS* implements using the LU decomposition approach.

Careful examination of code profiles show that the assembly phase and the LU decomposition account for over 90 % of the total execution time of large simulations. Consequently, we have concentrated the parallelization efforts on these two phases. A parallelization at a higher level of task granularity is out of the question since each iteration relies on the solution of the previous one to update the solution of the non-linear system. In the next sections, we describe the problems facing such a parallelization and discuss the approaches that we have chosen to implement.

# 3 Parallel Matrix Assembly

The assembly phase can be considered as a set of functions applied to each element of the mesh, evaluating the contribution of that single element to the overall structure. Considering the nature of the computations involved (associative operations), the order in which the elements are processed is irrelevant to the correctness of the results. In other words, they can be simultaneously executed. Consistently, *Parallel ATLAS* distributes the mesh elements responsibility to the available processors according to the results of a particular mesh partitioning technique. This mesh partitioning technique will assign neighboring elements to the same processor in order to reduce inter processor synchronization [1].

# 4 Parallel Sparse LU Decomposition

Efficient parallelization of sparse LU decomposition is still an active field of research. Good performance is usually extremely difficult to obtain and good scalability is even harder. The implementation in *Parallel ATLAS* features state of the art techniques developed at Silvaco, to achieve a very high level of efficiency on shared memory parallel supercomputers. We developed an event driven synchronization algorithm coupled to a level ordering of the task graph. This work is a generalization of the well documented sparse Cholesky factorization using elimination trees[2]. Further experiments indicate that the scalability of this implementation is sustained as the mesh size and the number of processors grow. The efficiency of the parallel code implemented in this phase is a key issue to the overall performance of the simulator.

# 5 Conclusion

The work described in this paper is the evidence that, thanks to parallelization, time can be less of a bottleneck to numerical device simulation. With viable execution times, there is less need for a trade-off between larger structures and finer details to achieve an acceptable accuracy in a reasonable amount of time. Table 1 show that *Parallel ATLAS* runs approximately 3 times faster on a 4 processors machine while simulating a wide variety of devices. Complete timing results are to be reported in the full paper.

| # processors | SOI | IGBT | ESD | HEMT | LATCHUP | CMOS |
|---|---|---|---|---|---|---|
| 1 CPU | 21h34mn | 7h20mn | 2h08mn | 1h36mn | 1h17mn | 0h28mn |
| 2 CPUs | 12h57mn | 4h22mn | 1h20mn | 0h57mn | 0h46mn | 0h17mn |
| 4 CPUs | 7h59mn | 2h27mn | 0h40mn | 0h35mn | 0h24mn | 0h10mn |

Table 1: *Parallel ATLAS* Execution times on a 4-CPUs SGI PowerChallenge.

# References


[1] Mounir Hahad, Thierry Priol, and Jocelyne Erhel. *Languages, Compilers and Run-Time Systems for Scalable Computers*, chapter Compilation of Assembly Patterns on a Shared Virtual Memory. Kluwer Academic Publishers, 1996.

[2] J.W.H. Liu. The role of elimination trees in sparse factorization. *Siam journal of matrix analysis applications*, pages 134–172, 1990.