Parallel and Distributed TCAD Simulations using Dynamic Load Balancing

R. Strasser and S. Selberherr

Institute for Microelectronics, TU Vienna
Gusshausstr. 27–29, A-1040 Vienna, Austria

Abstract

We present the job farming mechanism of the VISTA [1] TCAD framework, which enables efficient parallel and distributed simulation. A load balancing mechanism for optimal selection of computation hosts is introduced. Strategies for controlling the load of computation hosts are presented.

1. Introduction

From the technical point of view the simulation tools which are necessary to carry out rigorous device and process optimizations are available. Nevertheless, impractical simulation times prohibit large scale simulations. Therefore, maximum parallelism has to be exploited and the available computation resources have to be used optimally in order to reduce the overall simulation time to a minimum. Since simulation tools which are able to distribute themselves on several CPUs on several hosts are very rare and their communication overhead tends to degenerate their performance, a convenient alternative is parallelization of multiple instances of each simulation tool, as far as this is suitable. For many situations like optimization or split run experiments the latter approach offers excellent performance.

Fig. 1 shows the components involved in such a scenario. An optimizer is utilized to search for an optimum set of system parameters with respect to certain system characteristics. This typically requires the computation of Jacobian matrices, which needs large numbers of independent model evaluations that can be executed in parallel. The process model delivers its jobs to the *queue manager* which maintains two queues of waiting and executing jobs, respectively. The *queue manager* is constantly looking for hosts which are able to process these jobs while the system load of these hosts is periodically monitored.

2. Load Balancing

The *queue manager* is responsible for the efficient distribution of workload on a cluster of workstations, similar to existing queueing systems [2]. Therefore, computation hosts have to be registered with a performance metric w_i, the number of CPUs n_i^{cpu} and the desired maximum load l_i^{max}. To cope with simulation tools which require node locked licenses, these tools have to be registered defining the hosts at which

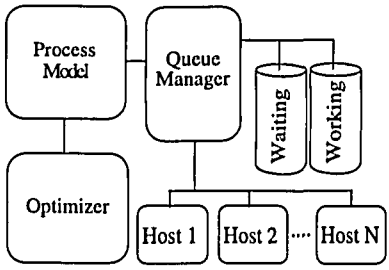

Figure 1: VISTA simulation architecture.

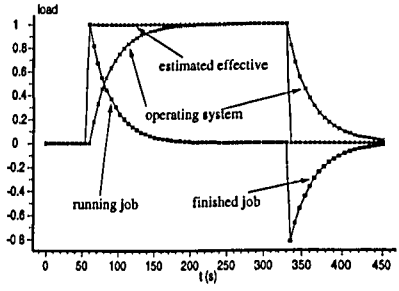

Figure 2: Running and finished jobs are used to estimate the effective load.

they are available. For each requested job the host which promises to finish the simulation within a minimum amount of time is selected, based on current load values, performance, and the number of CPUs. Thus the overall computation time is minimized on a heterogeneous cluster of workstations with different performance metrics. If a host exceeds its load limit, the host is automatically disabled. This mechanism guarantees an optimal usage of computation resources. Furthermore, adjustable load limits avoid conflicts among concurrent users.

In order to find the optimum host, it is necessary to make the decision based on an estimation of the current system load. Time delays of the load reported by the Unix operating system would lead to overloading of machines, since it typically takes five seconds up to one minute until a job is reflected in the system load. Therefore, VISTA keeps track of jobs that are executed by itself on each host and calculates an estimated effective load based on this information and the load reported by the Unix operating system. For a host with N jobs running and M jobs recently finished, the effective load is estimated by

$$l_{eff} = l_{os} + \underbrace{\sum_{i=1}^{N} e^{-\frac{t-t_i^{start}}{\tau}}}_{\text{running jobs}} - \underbrace{\sum_{i=1}^{M} \left[\left(1 - e^{-\frac{t_i^{stop}-t_i^{start}}{\tau}} \right) * e^{-\frac{t-t_i^{stop}}{\tau}} \right]}_{\text{finished jobs}}$$

as depicted in Fig. 2. In this formula τ is an estimate for the time by which the reported operating system load is delayed. The correction term for running jobs prevents hosts from being overloaded due to time delays. The contribution for finished jobs prevents hosts from being considered busy while the operating system still reports too much system load. Without such an estimation, both load balancing and load limit based disabling of hosts would give poor results. Either for large numbers of requested jobs a host would hopelessly be overloaded due to time delays, or if one limited the number of jobs on a host, inactive jobs, which are for some reason sleeping, would block hosts. On the other hand the operating system reports an unnecessary high system load for hosts which recently finished jobs, and therefore computation resources are unused until the system load drops down to the effective value. Fig. 3 shows a GUI to the *queue manager* where the status of each computation host is displayed in the *Hosts* section. Furthermore, queued and executing jobs are displayed in the *Queue* section.

3. Job Communication

Due to the parallel and asynchronous execution of system jobs sophisticated mechanisms to control these jobs and to communicate with them are necessary. Therefore, each job is represented by a VLISP `task` object which offers the following features: Input to a job is buffered as long as a job is in the waiting queue. This means that a client is able to send input to a job while it is still in a waiting state, and a job actually receives the input when a host becomes available and the job gets activated. Additionally the `task` object offers notification when the job is activated, output is available, the job failed, the job terminated, or the job succeeded. Clients of a job can subscribe these events and trigger specific actions by them. Thus full transparency is maintained, and job farming implies no restrictions for client applications exploring this functionality.

Fault tolerant mechanisms of job control are critical to large scale simulations since single point failures are likely to stop them. Especially for large numbers of workstations and heavy loaded networks such failures are very likely if the overall simulation time is in the range of several days. Therefore failed job executions are recognized and requeued and the concerned host is temporarily disabled. In practice, this mechanism has proven to be extremely effective and time saving since simulation experiments are not susceptible to single point failures and even survive disastrous situations like network failures.

4. Performance

To illustrate the benefits arising from parallelization let us consider the rigorous calibration of a device simulator. For the estimation of the required simulation time let us assume the following: Transfer curves (I_D/V_G) with the overall number of N operating points are available, M parameters have to be calibrated, I optimizer iterations are necessary, W workstations are available for computation, and the typical computation time required per operating point is T.

Given that each optimization iteration consists of gradient computation and evaluation, the overall computation time is roughly $(M+1) \times I \times T \times N$. Parallel evaluation of transfer curves reduces this time to $(M+1) \times I \times T \times \frac{N}{W}$. For $N = 30$, $M = 4$, $W = 15$, $I = 100$, and $T = 1\text{min}$, this means that VISTA job farming is able to reduce the time compared to operation on a single workstation from approximately 10 days to 16 hours.

5. Conclusion

A system for parallel distribution of computation workload has been presented. Using the VISTA *queue manager* as the front end for job execution in a TCAD framework ensures maximum computation performance and therefore reduces simulation times drastically which is of utmost importance, especially for extremely time consuming experiments like optimization and calibration. In fact, the VISTA *queue manager* enables optimizations which would be hopelessly out of a practicable time scale if they had to be carried out on a system with sequential execution of operating system jobs. Due to dynamic load balancing the user of the TCAD framework is deliberated from track the status of hosts and keeping their load below certain limits.

The VISTA *queue manager* is an integral part of the VISTA TCAD system and is therefore available on a wide range of Unix platforms including DIGITAL UNIX,

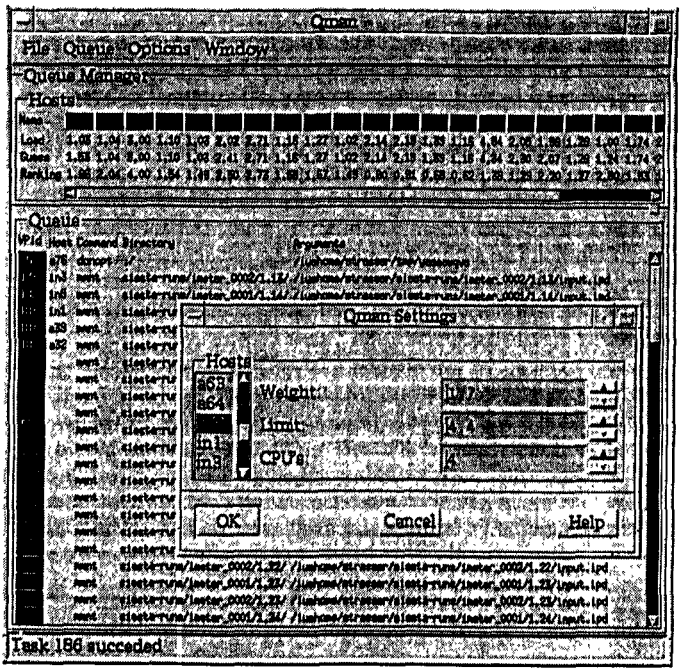

Figure 3: A GUI to the VISTA *queue manager* offers a comprehensive view of the status of computation hosts and it displays queued and executing jobs.

Solaris, HP-UX, Linux, IRIX, and AIX. Minimum prerequisites are necessary to use this system. Computers which are intended for collaboration have to share the users file system, and each user has to be able to remotely execute command via the `rsh` service. The system load is determined using the standard Unix program `uptime`.

References

[1] S. Halama, F. Fasching, C. Fischer, H. Kosina, E. Leitner, C. Pichler, H. Pimingstorfer, H. Puchner, G. Rieger, G. Schrom, T. Simlinger, M. Stiftinger, H. Stippel, E. Strasser, W. Tuppa, K. Wimmer, and S. Selberherr, "The Viennese Integrated System for Technology CAD Applications," in *Technology CAD Systems* (F. Fasching, S. Halama, and S. Selberherr, eds.), (Wien), pp. 197–236, Springer, 1993.

[2] Platform Computing Corporation, Toronto, Canada, *LSF Load Sharing Facility*, May 1998. `www.platform.com`.

Acknowledgment

This work was significantly supported by the "Christian Doppler Forschungsgesellschaft", Vienna, Austria and Austria Mikro Systeme, Unterpremstätten, Austria.