Evaluation of Parallel Logic Simulation Using DVSIM

Gerd Meister
Department of Computer Science
Technical University of Darmstadt
64283 Darmstadt, Germany
email: meister@informatik.th-darmstadt.de

Abstract

Parallel simulation is expected to speed up simulation run time in a significant way. This paper describes a framework that is used to evaluate the performance of parallel simulation algorithms. The framework's core is DVSIM, a parallel event-driven VHDL simulator. The framework provides several mechanisms to calculate sensible bases for speed-up calculation. Monitoring tools are employed to observe and to improve the algorithmic performance.

A first implementation of DVSIM used a conservative synchronization method, but a Time Warp protocol has recently been completed. Influencing factors for speed-up such as partitioning and mapping methods are discussed. Experience shows that even with conservative synchronization schemes moderate speed-ups can be obtained for larger circuits. The speed-up values are compared to theoretically possible acceleration factors, and the reasons why these ideal maximum speed-up values can in general not be reached are explained.

Keywords: Distributed Simulation, Load Balancing, Parallel Logic Simulation, Partitioning and Mapping, Speed-up, VLSI Design

1 Introduction

Simulation of VLSI circuits has become one of the major subtasks in the VLSI design process. It is used to detect or to avoid design errors and to verify functional correctness before the production of a chip starts. As designs increase in size and powerful parallel computers became generally available in recent years, the question naturally arose whether parallel simulation of VLSI circuits can effectively speed-up the simulation process. More specifically, one is also interested whether parallel simulation scales well and under which circumstances a significant decrease in simulation run time can be reached. One expects that the influencing factors are mainly the simulation principles and algorithms, the partitioning and mapping strategies, and design specific characteristics of the simulated circuits.

Although much work has already been done in the field of parallel logic simulation (Bailey et al. give a detailed overview and analysis [4]), the basic question whether reasonable speed-up of logic simulation using parallel machines is possible under general conditions has not yet been answered in a satisfying way. Until now, only techniques have been shown that are promising for some types of synchronization strategies and disappointing for others [10, 2, 22, 30]. On the other side, however, no statements were made that reasonable speed-up is generally impossible.

The published work, however, often examines single algorithms or specific details. Our framework allows to apply a variety of methods and investigates this issue on a broad basis. Even though only some representative algorithms can be examined in a single project, our selection should cover the whole spectrum of parallel discrete event simulation. Furthermore, the framework supports easy integration and implementation of new methods.

For logic simulation, two main simulation methods are in use: *synchronous time-driven simulation* [33, 3, 2] and *asynchronous event-driven simulation* [12, 24, 14]. The observed low activity rate[1] within synchronous time-driven simulation has led to the conclusion that the use of this simulation principle should in general be restricted to hardware supported simulations using special simulation processors (e.g., YSE [25] or MuSiC [15]). The second simulation method obeys the asynchronous event-driven paradigm. Here, specific events are associated with pertinent actions in the simulated system such as the change of a signal's value. While in the synchronous model all processors always operate at the same simulation time, in asynchronous event-driven simulation events for differing time steps

[1] The activity rate signifies the number of events that are scheduled for simultaneous evaluation, averaged over all points in simulation time.

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

(i.e., different points in simulation time) can in principle be evaluated independently by concurrently operating submodel simulators. These simulators, however, must be synchronized among each other to yield deterministic results and to maintain the correct execution order of the events.

Two main classes of synchronization strategies and protocols for parallel event-driven simulation have been reported in the literature: the *conservative* schemes [24] and the *optimistic* approaches [19].

The first class of strategies are somewhat overcautious. They avoid errors in the order of event execution by waiting for guarantees which permit them to be always on the safe side. Strategies from the second class (e.g., *Time Warp*) risk sequencing errors in order to be able to process events as soon as possible. In the case of an erroneous computation, the state of the simulated system is rolled back to a former state, and the execution order of the events is corrected. This requires periodically saved checkpoints of the state of the simulator. Beside these two basic categories, there exist a large number of variations, specific optimizations, and combinations of the basic strategies.

The usefulness of the basic schemes and the more specific variants has not yet been thoroughly examined and described in the context of parallel logic simulation. A general discussion of this issue can be found in [26]. Since it is not reasonable to assess each specific variant and to measure its potential for accelerating typical simulations, one has to resort to a faithful comparison of the main strategies in order to determine influencing factors. To our knowledge, such a broad and systematic comparison under realistic conditions has not yet been undertaken.

Our work is an approach to this problem. We implemented a parallel VLSI simulator that operates with the hardware description language VHDL [32] which became a de-facto standard in the last few years. The goal of the project is to examine the performance of different parallel simulation algorithms and to investigate whether in practice reasonable and scalable speed-up values are obtainable for parallel logic simulation.

The framework for the research work is sketched in the next section. In Section 3, the environment and the basic components of the parallel simulator are described in more detail. A rather important problem in parallel simulation is the mapping strategy. As in general there are less processors available than simulated objects, several objects must be mapped onto one processor. Measurements showed that the mapping strategy is of great importance for the performance of a parallel logic simulator (see also [4]). In Section 4, some mapping algorithms that we have implemented are compared and results from measurements are presented. We also present some conclusions comprising the current state of work and the findings from the results obtained from our experimental measurements. Some planned and currently implemented enhancements will be detailed and described in Section 5.

2 System overview

Our framework consists of the parallel and distributed logic simulator DVSIM, several sequential logic simulators which are instrumented for different speed-up calculations, and some trace and monitoring tools for the observation of DVSIM's behavior and performance.

DVSIM (**D**istributed **VHDL** **SIM**ulator) evolved from the sequential version VSIM developed at the University of Pittsburgh [21]. VSIM and DVSIM support a substantial subset of VHDL. VSIM was chosen because of its publicly available sources and to save us the need of a completely new and time-consuming development of a sequential simulator. Such a sequential simulator is necessary for the verification of the results of the parallel simulator and for faithful performance comparisons.

The parallel simulator was originally designed and developed on top of the message passing kernel MMK (Multiprocessor Multitasking Kernel) from the TOPSYS programming environment [9]. Recently, DVSIM was ported to the message passing library PVM [6]. The prototype simulator runs on both, a network of workstations and an Intel iPSC/860 hypercube [17]. In order to apply the asynchronous event-driven simulation technique, the whole circuit description is partitioned into disjoint parts. Each part is simulated by a dedicated so-called *logical process (LP)* [12]. Several LPs may be mapped onto one processor, however.

The goal of the development of DVSIM is twofold: firstly, to evaluate the scalability of parallel simulation on different hardware architectures and secondly to examine the behavior of different simulation synchronization algorithms. In addition to the basic simulation synchronization strategies, also adaptive and hybrid schemes will be investigated in the future. Adaptive algorithms may use application specific knowledge to improve the dynamic behavior of simulations. Hybrid schemes try to combine the benefits of conservative and optimistic methods.

To compare the scalability of parallel logic simulation on different hardware platforms, the MMK system and PVM were chosen because they run on several hardware types. The examination of different synchronization strategies and protocols is supported by using

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

the filter concept originally proposed by Reynolds [27]. It consists in encapsulating all protocol specific parts within a small number of functions. The exchange of one strategy against another is simply done by exchanging the corresponding functions. The protocol-independent code remains almost unchanged. This results in a modular system with easily maintainable code.

3 Implementation details

3.1 Programming environments

The MMK system. MMK is a flexible distributed programming system that allows the definition of different cooperating tasks, their mapping onto the nodes of a specified multiprocessor platform (workstations or hypercube), and the description of the communication structure between the tasks.

The different tasks of a distributed program are described in MMK in a function-like manner. The function parameters of such a description specify the interface for communication with other tasks. The body is either C or Fortran code and specifies the runtime behavior of the task. A preprocessor replaces each task header by C-code which controls the initialization and communication set-up of the tasks. All tasks of one application are compiled and linked into one executable binary that may be loaded onto a network of processors. A mapping file specifies for each processor which tasks will be activated.

Communication is performed through mailboxes that may be accessed via blocking and nonblocking primitives. Additionally, several MMK tasks may be loaded onto one processor. In the workstation environment, this is realized through a number of daemon processes that are running on each machine. They also control task creation and act as routers and managers for the communication layer. On the iPSC/860 hypercube, whose operating system only permits single-tasking, this daemon is compiled into the application code where it also acts as a scheduler for the MMK tasks on the processor.

PVM. As an effort to evaluate performance of parallel simulation on various other hardware platforms, DVSIM was ported to PVM. PVM provides basic facilities for task control and communication and became a de-facto standard for parallel programming in the last two years. The availability of PVM for many different architectures and also for parallel machines eases the problem of keeping software for a variety of such machines consistent and up to date. PVM increases the portability of code and allows fast and simple comparison of the suitability of different hardware architectures for parallel applications (such as parallel logic simulation).

While DVSIM has originally been developed under MMK, we switched to PVM as our preferred programming environment. MMK is, however, still supported to be able to compare the two environments. MMK has been very thoroughly tuned. Although both systems show the same qualitative behavior in measurements, the PVM version has approximately between 10 and 20 percent overhead compared to MMK. The main results presented in this paper were obtained using MMK.

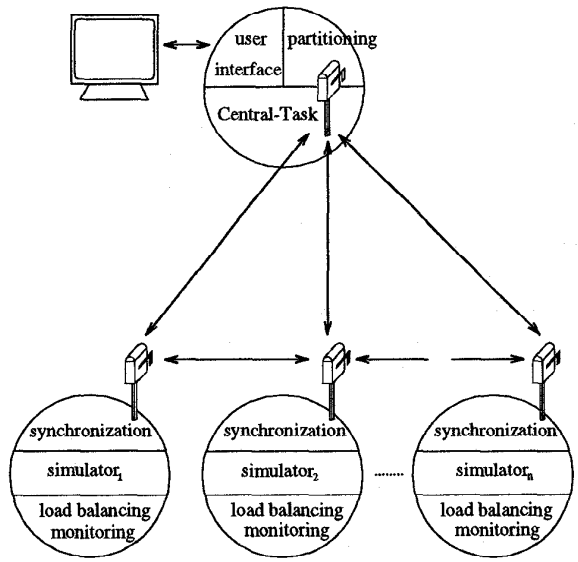

Figure 1: DVSIM system components.

3.2 DVSIM - basic features

The parallel and distributed logic simulator DVSIM consists of two main parts: a central task and one or several simulator tasks which provide the functionality of LPs (Figure 1). DVSIM uses an intermediate VHDL file (*ivf*) derived from a VHDL specification by compiling it with VCOMP [21]. VCOMP can only produce flat descriptions of the specified circuits at the gate level. This is a restriction of full VHDL which allows also hierarchical simulation of designs. In the following paragraphs, we will describe how DVSIM works.

Initialization. At the beginning of the simulation run, the programming environment (PVM or MMK) loads the central task and the simulator tasks onto the

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

specified configuration. Then, the central task reads the ivf-file and prompts for interactive input.

Partitioning. DVSIM provides several algorithms to map the circuit onto the available simulator tasks. Currently, we have implemented four methods:

- **Round-robin partitioning** distributes the gates in a circular way onto the available processors.
- **Acyclic partitioning** avoids feedback loops that cross partition borders. The circuit is viewed as a directed graph. The algorithm iteratively takes an output of the circuit and calculates its input cone (i.e., the transitive closure of preceding gates). The resulting gates form a cluster and are removed from the circuit. This procedure is repeated until all gates are assigned to a cluster. It ensures that cycles are contained in only one cluster. A cluster is entirely mapped onto one processor. The clusters are sorted according to their topological ordering. Then, the sorted clusters are assigned to the processors. The algorithm tries to keep the partition sizes (i.e., number of gates) equally balanced.

  Acyclic partitioning is important for conservative simulation strategies since it avoids communication deadlocks. Similar experiences are reported by Bagrodia et al. [1]. However, this method will produce poor results if a single or a few large cycles are contained in the circuit. Refinements which may avoid this disadvantage are described in [13].
- **Kernighan-Lin partitioning** is a modified bi-partitioning algorithm which reduces mainly the communication costs by iteratively exchanging pairs of elements between the partitions [20]. Each connection between two gates is assigned the initial cost '1'. The total costs are the sum of the costs from all connections that cross partition borders. Starting with a random partitioning, the algorithm picks sets of one or more elements from two partitions and exchanges the sets. If the modification reduces the total costs, the change becomes permanent. Otherwise, the original partitions are restored and the same procedure is repeated with other sets until no further improvement is found.
- **Soccer partitioning** [28] first builds a graph with one seed element for each LP. The elements of this graph are chosen in a way that the distance to all other elements is maximal concerning communication costs (using a modification of Dijkstra's Shortest Path Algorithm). Then additional elements are integrated into the basic partitions by assigning them to the partition to which they have the closest affinity (in terms of communication costs).

According to the chosen strategy, the central task determines the partitions and distributes the subsets of the circuit to the appropriate simulator tasks. Then the user may specify the simulation actions to be performed, such as setting of input signals, definition of output signals to be displayed, and starting the simulation for a predefined time interval.

For each interactive command, the central task determines the affected simulator tasks and sends an appropriate message to each of them. Upon completion of the simulation interval, a simulator task informs the central task about its termination. After the central task received such a confirmation from each simulator task, the input is returned to the interactive session.

Synchronization. Our first synchronization strategy for the simulator tasks followed a conservative approach [12]. The simulators execute only those events that are safe (i.e., where the corresponding guarantees are large enough to ensure that an event is executed in correct simulation time order). As is well-known, this behavior may result in global deadlocks which are caused by cyclic dependencies. In the case of a deadlock, a detection and resolution algorithm is initiated to continue simulation [24]. Recently, a synchronization strategy based on Time Warp [18, 19] has been implemented.

Monitoring. As we want to examine the behavior of different simulation algorithms, a monitoring component was included within the DVSIM simulator. This unit collects trace-data that describe the message-passing and event-execution related information as well as information concerning deadlock detection and *global virtual time*[2] [23] calculation. Trace-data is stored in event records. Each record comprises the type of the event, its simulation time stamp, and its physical execution time stamp. Of course, the probe effect from collecting trace-data must be minimized to prevent the instrumented system from showing a largely different behavior. For this reason, the data is first put into a local buffer and written to the file system only if either the simulator is blocked because of a deadlock, or if the buffer overflows (which is assumed

[2]Global virtual time (GVT) is a function of physical time which yields the minimum time stamp of all unprocessed events and messages in a simulation.

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

to be a rare event), or at the end of the simulation. Each LP writes its data to a dedicated trace-file.

3.3 Evaluation environment

The monitoring component allows the observation and tuning of parallel synchronization algorithms. As mentioned in Section 2, the synchronization strategies are encapsulated within filters which make them easily exchangeable. Together with the monitoring component this mechanism provides a framework to examine the suitability of different synchronization algorithms for parallel logic simulation. The generated trace-data is first preprocessed for the correction of the physical time stamps contained within the different trace-files. This is necessary because the local hardware clocks on the processors usually deviate and the recorded values have to be adapted. After this, the trace-files are sorted in physical time stamp order and merged into one single file. The resulting data serves as input to either the standard visualization package *ParaGraph* [16] or to a dedicated statistical event evaluator termed *YES*. While ParaGraph provides general views for parallel program visualization, YES allows viewing of different simulation specific aspects as event queue length, number of deadlocks, or depth of rollbacks.

These tools may help the user to understand the behavior of distributed simulation, to detect bottlenecks in the algorithms' implementations, and also to improve the overall performance.

3.4 Theoretical aspects of speed-up measurements

Speed-up measurement requires a base for comparison. Acceleration can be described only relative to such a base. An intuitive base is the run time of an optimized sequential program. Speed-up is the ratio between the run times of a sequential and a parallel simulation. This comparison, however, cannot tell enough about the efficiency of the examined parallel simulator. Efficiency denotes the ratio between the run times of an ideal parallel program and the parallel program under examination, for a given number of processors. Theoretically, an ideal parallel simulator which runs on n processors and which has no additional overhead for parallel execution should yield a speed-up of n and an efficiency value of 100 percent. In this section, three methods that deal with this issue are presented.

Critical path analysis on an ideal multiprocessor. Berry and Jefferson describe a technique to calculate the achievable speed-up of an ideal parallel simulation run [7]. The authors propose to measure the physical times needed for each event evaluation (including event-queue operations) in a sequential simulator. Using these measurements, the speed-up of an ideal parallel simulation compared to a sequential one is calculated. In the following, the term *real time* will be used as an equivalent to physical time.

The basic idea is to map each simulated object (the gates in our case) onto a dedicated processor which simulates only this object. Furthermore, it is assumed that each event is executed as early as possible (i.e., an event is evaluated if it is known and if all events for the corresponding gate with smaller time stamps have been executed). The communication latency for the propagation of events to remote gates (on other virtual processors) is assumed to be zero.

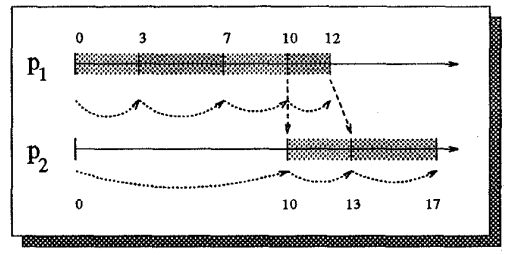

Figure 2: Speed-up calculation with CPA.

Berry and Jefferson's algorithm accumulates the time needed to evaluate each event on a per processor base. The simulation starts at time 0 for all processors. Each event carries a real time stamp. This stamp indicates the first point in physical time, the *start-time*, at which the event can be executed. After an event has been executed, its evaluation time is added to the value of the accumulated processor time. If new events are created by the current event, their start-time is set to the maximum value of the accumulated time of the creating processor and the accumulated time of the processor for which the event was created. The start-time is the first point at which the new event can be evaluated in an ideal parallel simulation. If a remote event for an idle processor is created, the processor time is artificially raised to the start-time of this event before it is executed (Figure 2).

Additionally, the execution times for all events are summed up, yielding the value t_{seq}. At the end of the simulation run, the maximum of all accumulated processor times, t_{par}, represents a lower bound on the run time needed by any (conservatively controlled) parallel simulator. This value is the end point of the critical path that leads through the space-time diagram of the simulation run. Note that all calculations are

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

performed *on the fly* by a sequential simulator. The ratio

$$speed\_up_{cpa} = \frac{t_{seq}}{t_{par}}$$

is a measure for the obtainable speed-up. In the simple example of Figure 2, t_{seq} is 19 and t_{par} is 17 yielding a speed-up of 1.12. Because of the use of the critical path, this method is called *critical path analysis* (CPA). However, using an unlimited number of processors for the approximation is merely of theoretical interest because of the restricted number of processors of parallel machines.

CPA on a multiprocessor with limited resources. Because of the limited number of processors, a variant of the algorithm where multiple gates are assigned to one processor according to a given mapping is more sensible. Here, the accumulated processor time is increased by the execution times of all events that are simulated by one processor. The realization of this algorithm is straightforward. The speed-up measurement is calculated as before and yields a statement on the speed-up with limited resources for a given mapping strategy.

Oracle log. To get a more realistic estimate on the obtainable speed-up, the communication overhead and message latencies should be taken into consideration. This problem is focused by Swope and Fujimoto [31]. They propose a method called *oracle log*.

The oracle log is produced (for a given mapping) by a conventional sequential or parallel simulator. Whenever a signal that crosses partitions changes its value, it will be registered and a log is written to the logfile. Each log consists of the simulation time stamp of the event, the new value and the identifier of the signal.

In a second (parallel) run, the modified synchronization algorithm simply asks the oracle whether the simulation may proceed or not instead of waiting for guarantees as conservative simulation strategies usually have to. The simulator only blocks if the oracle tells that there are outstanding events from another LP.

While CPA is based on a pure sequential simulation, the oracle log method runs as parallel simulator. The ratio between the time needed by the sequential simulation and by the parallel oracle log simulation is a measure for the obtainable speed-up under consideration of communication overhead but assuming an ideal synchronization protocol.

Because an additional simulation run is necessary to produce the oracle, oracle log is of course not a practical simulation scheme, which may be used by a circuit developer. Instead, CPA and oracle log are used during development of parallel synchronization algorithms to assess the quality of the new strategies and to determine the effect of message overheads and synchronization costs of hardware platforms.

4 Measurement results

4.1 The impact of partitioning and mapping

DVSIM is realized on top of message passing architectures. However, the exchange of a message is often rather expensive — during one send or receive operation, multiple event evaluations can usually be performed (at least at the gate level). One major aspect of partitioning is to keep communication cost low (i.e., the need to send messages to other processors should be reduced). Another aspect that poses problems (at least for many conservative synchronization strategies) is the existence of feedback loops within circuits. For conservative synchronization schemes, guarantees grow very slowly within a cycle. Only few simulators have sufficient information to execute events safely. If the lookahead [14] is zero, deadlocks will occur frequently within the system. This becomes even worse if the objects (e.g., gates) in such a cycle are located on different processors.

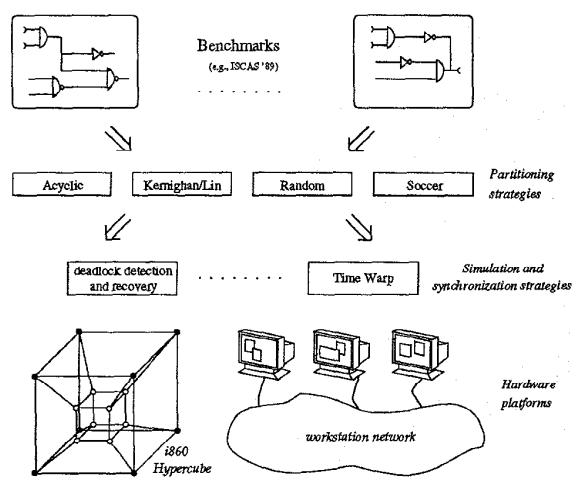

Figure 3: Environment for speed-up measurements.

Placing whole cycles on one processor increases the local knowledge on causal dependencies and consequently more events of the involved objects may be safely processed. Expensive messages that are needed to resolve deadlocks are thus avoided.

We implemented four mapping algorithms. Two of them try to minimize communication costs: Ker-

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

nighan-Lin and Soccer. Feedback loops are avoided by the third method, an acyclic partitioning scheme. Finally, the round robin method serves as a lower bound for the quality of any other partitioning scheme.

Measuremensts which emphasize the impact of partitioning and mapping on the obtainable speed-up will be shown in the next subsections.

4.2 Conservative simulation using different partitioning methods

Using a conservative synchronization strategy, we simulated three different sequential circuits from the well-known ISCAS89 benchmark suite [5]: s1196, s13207, and s35932. In the VHDL specification where the D-flipflops are converted to the corresponding gate representation, they consist of 892, 15709, and 40685 gates. Figure 3 shows the environment for the measurements.

As input, we supplied random stimuli vectors of 32 bit length to each external input of the circuits. The resolution of simulation time was 0.1 ns, and stimuli values were assigned to the input signals every microsecond[3]. DVSIM was executed under MMK on 1, 2, 4, 8, 12, and 16 processors of an Intel iPSC/860 hypercube.

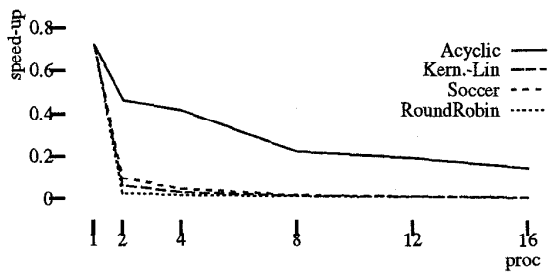

Figure 4: Speed-up of benchmark s1196.

Figures 4 to 6 show the speed-up that results from the experiments. The ratio between sequential and parallel run time yields the speed-up values displayed in the figures. It should be noted that the small circuit s1196 (Figure 4) has no speed-up at all. A speed-up value less than 1 for DVSIM on only one processor indicates that the distributed simulator contains some inherent overhead due to more complex algorithms compared to the sequential case. The overhead is, however, small enough to allow a fair comparison between the sequential and parallel simulator.

The graphs show that the performance of distributed simulation not only depends on the specific circuit. It is also strongly influenced by the mapping scheme that is

[3]Of course, two successive assignments may have the same value.

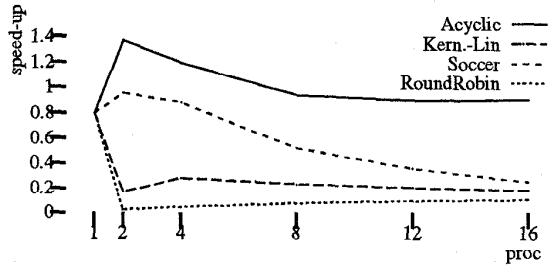

Figure 5: Speed-up of benchmark s13207.

used. Obviously, the acyclic partitioning scheme provides very good results compared to the other ones. The reasons for this are:

- Usage of an acyclic partitioning scheme avoids deadlocks that may appear in a conservatively synchronized simulation. For this reason, less work has to be done within the deadlock resolution algorithm and less messages must be sent to propagate information on safe events.
- An acyclic partitioning scheme keeps neighboring elements together on one processor (at least if they belong to the same loop). This again reduces the number of messages that will be sent during simulation. As posting messages is expensive on many distributed system platforms, this again speeds up simulation.

Another important observation is that there is no speed-up at all for smaller simulation models. As shown in Figure 4, the sequential simulator outperforms the distributed simulator by far. For the larger benchmark s13207 (Figure 5) already some true speed-up is observed for the acyclic mapping strategy. For s35932 (Figure 6), which has more than twice the size of s13207, the speed-up factors further increase.

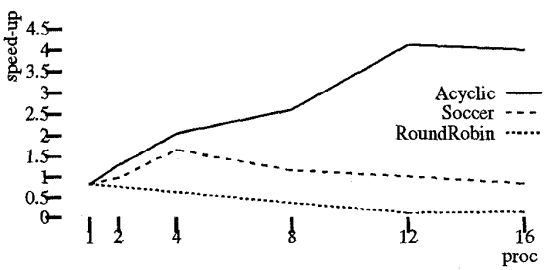

Figure 6: Speed-up of benchmark s35932.

For the larger benchmark circuits, one also observes saturation for an increasing number of processors. Although more processors are used, speed-up stagnates

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

or even decreases. This is caused by the fact that increasing the number of processors also reduces the sizes of the partitions for a given circuit. If the number of objects per simulator is lower than a certain level, there is simply not enough work and run time is dominated by communication and overhead for the parallel algorithms. Our investigations show that it is not sensible to have partition sizes less than four or five thousand gates.

4.3 Measuring the potential parallelism

The results of the previous subsetion do not show the naively expected linearly increasing speed-up curves, but they let one hope that for larger models the performance could be further improved. This is also supported by the measurements from critical path analysis (Figure 7) which show that there is still more exploitable parallelism available from the benchmark circuits. Therefore, if one can improve the ratio between computation and communication (e.g., by using better partitioning strategies or other synchronization mechanisms), the simulation is likely to run faster than our preliminary experiments indicate.

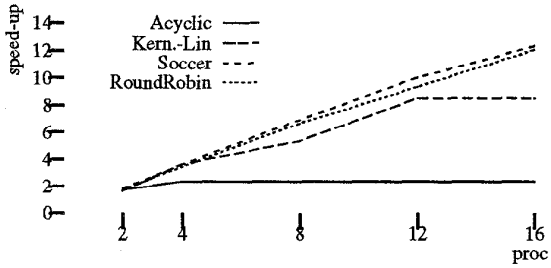

Figure 7: CPA speed-up of benchmark s13207.

Interestingly, the curve for acyclic partitioning in Figure 7 remains nearly constant for an increasing number of processors. During acyclic partitioning, the mapping algorithm tries to keep the number of gates balanced among all processors. As circuit s13207 contains a very large cycle, this strategy fails for a larger number of processors. The run time of the whole simulation is dominated by the time needed to simulate this cycle. This example shows that simple partitioning schemes sometimes produce inadequate results. Schemes which consider the structure of circuits might be promising.

Figure 8 compares the performance of the conservative and of the oracle log method for circuit s13207. The differences between the speed-up values of the oracle log and the conservative simulation are a measure for the overhead that is produced by unnecessarily blocking in conservative simulation. The differences

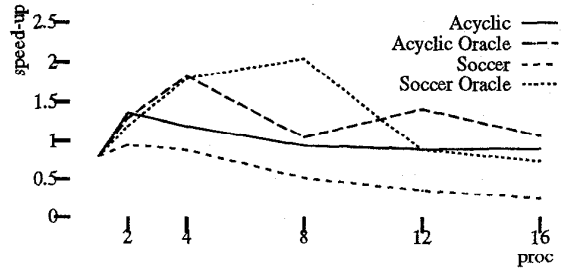

Figure 8: Speed-up of s13207: oracle vs. conservative.

for the acyclic partitioning scheme are small because blocking is infrequent.

Using soccer partitioning, the differences are much larger. Although oracle log performs sometimes better with soccer than with acyclic partitioning, this is not true for the conservative simulation method. The reasons are as following. Firstly, soccer partitioning generally yields better load balance than the acyclic method in terms of the number of gates. Since unnecessary blocking is avoided with the oracle log method, this will result in shorter run times. Secondly, the number of messages that are sent increases under soccer, and cycles that cross partition borders are not avoided. This causes additional deadlocks and blockings for the conservative strategy.

4.4 Preliminary results using Time Warp

Figure 9 depicts preliminary results using the optimistic Time Warp strategy for circuit s13207. The measurements were performed on a network of workstations under PVM. It shows that for larger numbers of processors, Time Warp with soccer partitioning yields shorter execution times (i.e., better speed-up values) than the conservative simulation.

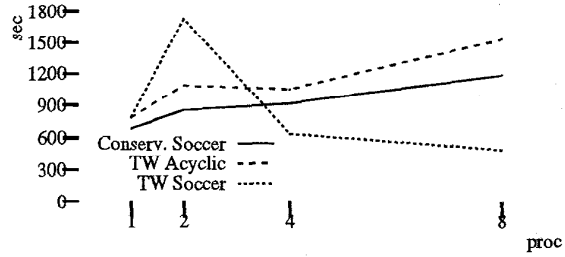

Figure 9: Time Warp vs. conservative run time of s13207 on workstations.

Interestingly, the acyclic partitioning scheme which is well suited for conservative simulation performs poorly with Time Warp. This can be explained as follows: When four or more processors are used for Time

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

Warp, circuit s13207 is badly balanced by the acyclic method and the LPs operate almost independently of each other. They proceed very fast in simulation time and are penalized for their optimism with frequent rollbacks. With the soccer method the causal dependencies among the LPs increase, and the local clocks are prevented from drifting too far. Rollback costs are reduced resulting in better performance. This shows that Time Warp offers many possibilities for variants (such as time windows) and optimizations.

5 Conclusions and future work

The overall goal of our project is to investigate the conditions that lead to reasonable speed-up of logic simulation using parallel architectures.

As the previous section revealed, partitioning plays an important role in parallel logic simulation's performance. With a good mapping scheme, already with conservative synchronization algorithms speed-up over sequential simulation can be achieved for larger circuits. For optimistic strategies, we observed encouraging tendencies. Similar results are reported in the literature [8, 22].

We presented results for three real benchmark circuits. Of course, the investigations will have to be extended to other benchmarks. Also the stability of the behavior with respect to the provided stimuli must be further examined.

The comparison of the obtained speed-up values from parallel simulation to the results of CPA revealed that although some speed-up was observed in the experiments, the maximum CPA values are not reached. One main reason is the large overhead caused by communication which must not be neglected.

Our current work focuses on the optimization of Time Warp, the application of the oracle log method to examine and to improve the conservative algorithms, and the integration of additional partitioning schemes that use application specific knowledge (e.g., strings and cone partitioning) [29]. Additionally, conservative and optimistic methods will be combined to hybrid schemes, compared, and tuned using our trace-based analysis and visualization tools.

Finally, we are implementing a dynamic load balancing scheme. This makes sense because in static partitioning schemes the criteria for placement of single gates have to be estimated before the actual behavior during the simulation is known. Our current approach to this problem is to run the simulator for a short time, accumulate data on the number of event evaluations and data on the signal activities, and to use this data for another iteration of partitioning (so-called *pre-simulation*) [11]. This method may improve the performance. However, for long running simulations the estimates from the initial phase might not be sufficient. One could repeat the partitioning scheme several times during the simulation for further improvement. This, however, might be expensive, especially for large circuits. Therefore, we expect further gains from dynamic load balancing if it is realized efficiently.

As a final remark, it should be noted that the results presented in Section 4 contained only the plain simulation times. If one considers the speed-up of a simulation session, the costs for running the partitioning algorithm, for initialization of the parallel simulator, and for the distribution of the gates onto the processors have also to be accounted. These factors severely limit the overall performance of parallel simulation in general. They must still be improved in the current prototype version of DVSIM.

References

[1] BAGRODIA, R., LI, Z., JHA, V., CHEN, Y., and CONG, J. (1994) *Parallel Logic Level Simulation of VLSI Circuits.* Proc. of the Winter Simulation Conference, pp. 1354 – 1361

[2] BAILEY, M. (1992) *How Circuit Size Affects Parallelism.* IEEE Trans. on Computer-Aided Design, Vol. 11, pp. 208 – 215

[3] BAILEY, M. and SNYDER, L. (1988) *An Empirical Study of On-chip Parallelism.* Proc. of the 25th Design Automation Conference, ACM/IEEE, pp. 160 – 165

[4] BAILEY, M., BRINER, JR., J., and CHAMBERLAIN, R. (1994) *Parallel Logic Simulation of VLSI Systems.* Computing Surveys, Vol. 24, pp. 255 - 294

[5] BRGLEZ, F., BRYAN, D., and KOZMINSKI, K. (1989) *Combinational Profiles of Sequential Circuits.* Proc. of the ISCAS'89, pp. 1929 – 1934

[6] BEGUELIN, A., DONGARRA, J., GEIST, A., MANCHEK, R., and SUNDERAM, V. (1992) *A User's Guide to PVM.* Technical Report ORNL/TM-11826, Oak Ridge National Laboratory

[7] BERRY, O. and JEFFERSON, D. (1985) *Critical Path Analysis of Distributed Simulation.* Proc. of the Distributed Simulation Conference, San Diego, pp. 57 – 60

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE

[8] BRINER, JR., J., ELLIS, J., and KEDEM, G. (1991) *Breaking the Barrier of Parallel Simulation of Digital Systems.* Proc. of the 28th Design Automation Conference, ACM/IEEE, pp. 223 – 226

[9] BEMMERL, T. and LUDWIG, T. (1990) *MMK – A Distributed Operation System Kernel with Integrated Loadbalancing.* Proc. of the CONPAR 90 VAPP IV, Springer-Verlag, LNCS 457, pp. 744 – 755

[10] BRINER, JR., J. (1990) *Parallel Mixed-Level Simulation of Digital Circuits Using Virtual Time.* PhD thesis, Duke University, Technical Report TR90-38

[11] CHAMBERLAIN, R. and HENDERSON, C. (1994) *Evaluating the Use of Pre-Simulation in VLSI Circuit Partitioning.* Proc. of the PADS Workshop, pp. 139 – 146

[12] CHANDY, K. and MISRA, J. (1981) *Asynchronous Distributed Simulation Via a Sequence of Parallel Computations.* Comm. of the ACM, Vol. 24, pp. 198 – 206

[13] CONG, J., LI, Z., and BAGRODIA, R. (1994) *Acyclic Multi-Way Partitioning of Boolean Networks.* Proc. of the 31st Design Automation Conference, ACM/IEEE, pp. 670 – 675

[14] FUJIMOTO, R. (1990) *Parallel Discrete Event Simulation.* Comm. of the ACM, Vol. 33, pp. 30 – 53

[15] HAHN, W. and FISCHER, K. (1985) *An Event-Flow Computer for Fast Simulation of Digital Systems.* Proc. of the 22nd Design Automation Conference, ACM/IEEE, pp. 338 – 344

[16] HEATH, M. and ETHERIDGE FINGER, J. (1991) *Visualizing Performance of Parallel Programms.* Technical Report ORNL/TM-11813, Oak Ridge National Laboratory

[17] *Intel iPSC/2 and iPSC/860 User's Guide.* (1991) Intel Cooperation

[18] JEFFERSON, D. (1985) *Virtual Time.* ACM Transactions on Programming Languages and Systems, Vol. 7, pp. 404 – 425

[19] JEFFERSON, D. and SOWIZRAL, H. (1985) *Fast Concurrent Simulation Using the Time Warp Mechanism.* Proc. of the Conference on Distributed Simulation, pp. 63 – 69

[20] KERNIGHAN, B. and LIN, S. (1970) *An Efficient Heuristic Procedure for Partitioning Graphs.* Bell System Technical Journal, Vol. 49, pp. 291 – 307

[21] LEVITAN, S. (1993) *VCOMP & VSIM Reference Manual, Edition 1.2.1.* University of Pittsburgh

[22] MATSUMOTO, Y. and TAKI, K. (1992) *Parallel Logic Simulation on a Distributed Memory Machine.* Proc. of the European Conference on Design Automation, pp. 76 – 80

[23] MATTERN, F. (1993) *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation.* Journal of Parallel and Distributed Computing, Vol. 18, pp. 423 – 434

[24] MISRA, J. (1986) *Distributed Discrete-Event Simulation.* Computing Surveys, Vol. 18, pp. 39 – 65

[25] PFISTER, G. (1982) *The Yorktown Simulation Engine: Introduction.* Proc. of the 19th Design Automation Conference, ACM/IEEE, pp. 51 – 54

[26] REYNOLDS, P. and DICKENS, P. (1989) *SPECTRUM: A Parallel Simulation Testbed.* Hypercube Conference

[27] REYNOLDS, P. (1988) *A Spectrum of Options for Parallel Simulation.* Proc. of the Winter Simulation Conference, pp. 325 – 332

[28] RUNO, D. (1988) *Das Graphpartitionierungsproblem und seine Lösungsmethoden.* Master's thesis, GMD, Bonn

[29] SMITH, S., MERCER, M., and UNDERWOOD, B. (1987) *An Analysis of Several Approaches to Circuit Partitioning for Parallel Logic Simulation.* Proc. of Int. Conference on Computer Design, IEEE, pp. 664 – 667

[30] SOULÉ, L. (1992) *Parallel Logic Simulation: An Evaluation of Centralized-Time and Distributed-Time Algorithms.* PhD thesis, Stanford University, Technical Report CSL-TR-92-527

[31] SWOPE, S. and FUJIMOTO, R. (1987) *Optimal Performance of Distributed Simulation Programs.* Proc. of the Winter Simulation Conference, pp. 612 – 617

[32] *IEEE Standard VHDL Language Reference Manual.* (1987) IEEE Std. 1076-1987

[33] WONG, K., FRANKLIN, M., CHAMBERLAIN, R., and SHING, B. (1986) *Statistics on Logic Simulation.* Proc. of the 23rd Design Automation Conference, ACM/IEEE, pp. 13 – 19

Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE