Parallel Timing Simulation on a Distributed Memory Multiprocessor [*]

Chih-Po Wen Katherine A. Yelick
Computer Science Division
University of California, Berkeley, CA 94720

Abstract

We present a parallel timing simulator, PARSWEC, that exploits speculative parallelism and runs on a distributed memory multiprocessor. It is based on an event-driven timing simulator called SWEC. Our approach uses *optimistic* scheduling to take advantage of the latency of digital signals. Using data from trace-driven analysis, we demonstrate that optimistic scheduling exploits more parallelism than conservative scheduling for circuits with feedback signal paths. We then describe the PARSWEC implementation and discuss several design trade-offs. Speedups over SWEC on large circuits are as high as 55 on a 64-node CM5 multiprocessor. These results indicate the feasibility of using distributed memory multiprocessors for large-scale circuit simulation.

1 Introduction

We present a parallel timing simulator, PARSWEC, developed for distributed memory multiprocessors. PARSWEC is a parallelization of SWEC [1], an event-driven timing simulator. SWEC employs a stepwise linear waveform and device model, in which subcircuits are evaluated by solving a linear system of node voltages. The evaluation of a subcircuit is triggered by *events* generated by the state changes of the subcircuit and its fanins. The rate of state change determines the time step size to be used for a given subcircuit at a particular time point. These features take advantage of the *latency* and *multirate* properties in most digital circuits [2]. The latency property states that most digital signals change infrequently; the multirate property states that different parts of a circuit produce signals at different speeds.

The PARSWEC algorithm partitions a circuit into loosely coupled subcircuits and then assigns those subcircuits to processors. Each processor is responsible for simulating the subcircuits stored in its local memory. The subcircuits are simulated *optimistically*, by assuming that no events will be generated by their fanins. Optimistic scheduling exploits the parallelism determined by the actual signal flows at runtime, rather than limiting parallelism to the static interconnection of the circuit. A previous study shows the lack of static parallelism for several CMOS circuits [3]. We demonstrate that for sequential circuits, available run-time parallelism is much higher than static parallelism. This justifies our choice of optimistic scheduling—although it requires more memory and potentially more total computation than conservative scheduling, it leads to better overall performance on a large scale multiprocessor.

In this paper we present the following:

- trace-driven simulation results that demonstrate the parallelism advantage of optimistic scheduling for sequential CMOS circuits;
- a parallel timing simulation algorithm based on the Timewarp idea of Jefferson [4], which was previously used for other simulation domains;
- an implementation for a scalable (distributed memory) multiprocessor, which is highly tuned for locality to maximize performance;
- analysis of the design trade-offs and performance costs in our implementation.

2 Problem Statement

The first step in a timing simulation is to partition the circuit based on the charge coupling of transistors. In digital MOS circuits, two transistors are tightly coupled if they are connected via a source-drain charge path, and loosely coupled if they are connected only via the gate terminal of some transistor. Tightly coupled transistors must be evaluated simultaneously for precision, while loosely coupled transistors can be evaluated mostly independently, with infrequent propagation of values between transistors. We refer to the clusters of tightly coupled transistors as *subcircuits*.

[*]This project was supported in part by the Semiconductor Research Consortium, by the National Science Foundation (grant numbers CCR-9210260 and CDA-8722788), by Lawrence Livermore National Laboratory, by the Advanced Research Projects Agency of the Department of Defense under contract DABT63-92-C-0026, and by AT&T.

In SWEC, voltage waveforms are modeled by a piecewise linear approximation. Time steps may vary across time and between different subcircuits, and have a minimum value of one picosecond. Time steps are adjusted so that input and output waveforms of all subcircuits are linear within a small error margin. Several time steps may elapse before a subcircuit communicates its state because a subcircuit propagates its output only when the new output cannot be linearly extrapolated using the old one.

A subcircuit is represented by a data structure called a **region**, for which a single time step simulation is called a *region evaluation*. In SWEC, a region evaluation involves model evaluation and a linear system solve using Gaussian elimination or relaxation. A region evaluation leads to an *event* if the new state is not a linear approximation of the old state. An event is the communication of new states to the fanout regions.

To maintain the causality of evaluations, SWEC uses a priority queue of regions ordered by the *next evaluation time* of each region. After a region is evaluated, the next evaluation time is predicted assuming that no fanin events will be generated. If fanin events are generated, the next evaluation time is adjusted, leading to a different ordering in the priority queue. Since events always propagate forward in time, the region at the head of the queue is guaranteed to be ready for evaluation.

3 Discrete Event Simulation

Discrete event-driven simulation is used to simulate a physical system as a collection of processes, in our case processes to evaluate subcircuits. Processes communicate by sending events, which are time-stamped messages containing state values. Two techniques for parallel discrete event simulation correspond to two different scheduling policies. The Chandy-Misra algorithm uses *conservative* scheduling [5], and the Timewarp algorithm uses *optimistic* scheduling.

In the Chandy-Misra algorithm, each process keeps a logical clock to denote its progress. Progress is conveyed to other processes via the timestamps of events. The conservative algorithm schedules process p for evaluation at time t only when all processes that may send an event to p have been simulated past t. Deadlocks can occur if there are cyclic dependencies in between processes, so deadlock prevention or detection is needed. Both of these require global information and therefore incur communication and synchronization overhead in a distributed environment. In logic simulation, for example, the Splash study shows that deadlocks severely limit parallelism [6].

The Timewarp algorithm relies on the rarity of events. Processes may compute ahead even if more events for earlier times are forthcoming. If events do come, the results are discarded and evaluation resumed from a previous time point. This operation is known as a *rollback*, and the event causing the rollback is called a *straggler*. To restart after a rollback, each region must maintain a *history* of received events and its own past states. A process that rolls back may have to send *anti-messages*, whose sole purpose is to undo the effect of its invalid events. Note that invalid events may be caused directly by a straggler or indirectly from anti-messages caused by another straggler of another process. The optimistic approach requires a large amount of memory for storing history information and incurs overhead from rollbacks, but as we show in Section 4, it yields more parallelism than a conservative approach for timing simulation of sequential circuits.

4 Measuring Available Parallelism

Bailey and Snyder performed a study of available parallelism in digital circuits [3] to explain the limitations in parallelizing SPICE-like circuit simulators. They measured the *real time parallelism*, the average number of transistors switching at the same time, in six circuits. The results showed low parallelism even for large circuits, for example, only a factor of 6.3 for a 32-bit RISC processor containing over 24,000 transistors. This real time metric is a limit for synchronous parallel simulators, where subcircuit evaluations are kept synchronized to the same real time, but asynchronous timing simulators may proceed in parallel as long as dependencies are preserved, so the limits do not apply.

We give the results of a similar study, but measure parallelism on the simulation time axis, rather than the real time axis. Specifically, we evaluate the effectiveness of conservative and optimistic scheduling applied to SWEC simulation. Our results are far more encouraging, particularly for the optimistic approach.

Figure 1 lists some characteristics of our benchmark circuits. In this section, we use only some of the smaller circuits, since the time to compute parallelism profiles for large benchmark circuits would be prohibitive. The 16-bit ripple adder (ADDER) and the two 16-bit multipliers (MUL1 and MUL2) are combinational circuits. The register file (REGFILE) is sequential, but there are no feedback paths among the regions because all nodes on a feedback path are tightly coupled, and are partitioned into the same region. The 4-bit counter (COUNTER) consists of 4 T type flipflops. The 4-bit counter, the PLA state

Name	mosfets	regions	time
ADDER	442	129	59
MUL 1	7190	401	293
MUL 2	6234	1101	5297
COUNTER	170	51	22
PLA	2117	507	1423
REGFILE	4832	404	538
SIMD	37939	7413	64916
C1355	2306	678	942
C2670	5364	2033	5919
C5315	11260	3730	21731
C7552	15394	5272	43086

Figure 1: Benchmark circuits: the last column is running time in seconds of sequential SWEC on a Sun/4.

Combinational	ADDER	MUL 1	MUL 2
Conservative	40.6	67.3	107.7
Optimistic	40.9	67.4	109.5
Sequential	COUNTER	PLA	REGFILE
Conservative	2.5	1.9	27.6
Optimistic	8.3	21.5	27.6

Figure 2: Parallelism in the benchmark circuits.

machine (PLA), and the SIMD processor datapath (SIMD) are sequential circuits with many feedback paths among the regions. C1355, C2670, C5315, and C7552 are drawn from the ISCAS benchmark suite.

We instrumented sequential SWEC to record the activities of the simulation. To give an upper bound on available parallelism, we assume that there are an unlimited number of processors, that communication is free, and that each evaluation is assigned to an idle processor as soon as it is "ready," as defined by either optimistic or conservative scheduling. For optimistic scheduling, we assume that there is an oracle to predict the presence of events, so the cost of rollbacks is not counted. To model the computational cost of a region evaluation, we use its average running time on a CM5 (Sparc) processor. A plot of the number of busy processors over time gives the parallelism profile over the entire simulation. The *average parallelism*, is then the ratio of the sequential running time and the parallel running time.

Figure 2 shows the conservative and optimistic parallelism for some of the benchmark circuits. In all of the circuits there is a burst of activity at the beginning of the simulation, when the circuit is converging to a stable (DC) state. Therefore, the circuits are simulated long enough so that the start-up effect is not

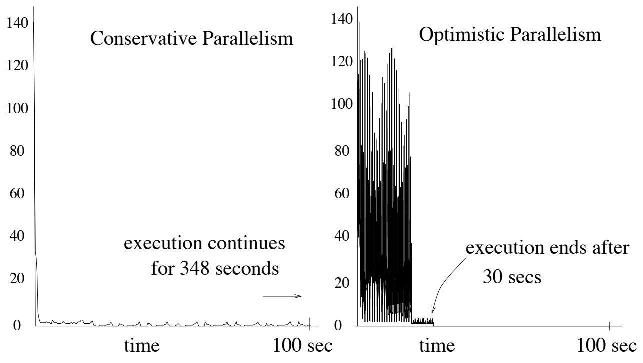

Figure 3: Circuit parallelism profiles: degree of parallelism over time.

significant. A striking advantage of optimism is seen for the PLA circuit. The parallelism profile for PLA in Figure 3 shows this advantage more graphically.

Our results show that parallelism usually grows with the number of regions and that optimism has only minor effect on combinational circuits, but a significant impact on circuits with feedback paths. In combinational circuits, both scheduling policies allow dependent regions to progress in a *pipelined* fashion, so the total running time is dominated by the critical paths in the circuits. In sequential circuits, conservative scheduling cannot pipeline evaluations along a feedback cycle: *No two regions in a feedback path can be evaluated in parallel.*[1] In contrast, optimistic scheduling exploits pipelining along cycles to the extent limited by the critical path.

The advantage of optimistic scheduling is quantified by in Figure 2. The PLA state machine shows an 11-fold increase in parallelism and the counter shows a 3-fold increase. Although we were unable to measure the SIMD processor datapath or the other larger benchmark circuits, the bus structure of processors like SIMD will certainly prevent the circuit from exhibiting high parallelism with conservative scheduling.

Under optimistic scheduling, if two regions in a feedback path are evaluated in parallel, the outcome of one may render the other incorrect. However, the latency property of digital circuits indicates this is not the common case. In the SIMD circuit, for example, a bus-oriented interconnection makes most regions of the circuit mutually dependent, but the actual signal flow is carefully controlled by the timing scheme; most of the time the circuit is acting as a collection of independent functional units, whose signals are confined

[1]Technically, under conservative scheduling, two regions can be evaluated in parallel if they happen to have identical simulation times, but because the time-steps are chosen dynamically and independently, this is rarely, if ever, the case in practice.

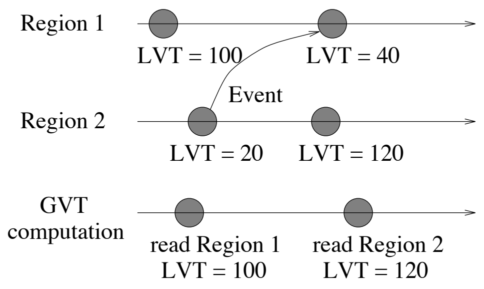

Figure 4: Computing GVT: The value is 40, but the naive scheme gives 100.

to its own latches or registers.[2] Our performance results in Section 6 confirm that the degree of optimistic parallelism in SIMD and other large circuits is high.

5 The PARSWEC Implementation

In optimistic simulations, the progress of a region is represented by the timestamp of its most recent state, referred to as the *local virtual time* or LVT. The progress of the system is determined by the minimum of the timestamps of all regions and messages, called the *global virtual time* or GVT. The LVT of individual regions may be decreased by stragglers or anti-messages, but the GVT of the system is monotonically increasing. GVT is used for *fossil collection* and *termination detection*. Fossil collection discards states with time less than GVT, since no region can be rolled back before GVT. Termination detection simply halts the simulation when GVT progresses past the final simulation time.

Computing GVT requires taking a snapshot of the distributed state. A naive implementation that probes the processors individually would be incorrect, since, as shown in Figure 4, rollbacks may be missed. Distributed snapshot algorithms exist, but they add message overhead on communication. Because there is hardware on the CM5 for fast synchronous operations, we use a synchronous algorithm based on *fuzzy barriers*, in which processes initiate a request to perform a barrier, and then continue computing until all other processors agree.

Each circuit region contains a *state history* of previous region states, and an *event history* of previous fanin events. In addition, a region has a FIFO message queue for each fanout edge, which stores unprocessed event messages and anti-messages. A processor has a set of these regions, and a local copy of the GVT variable, initialized to 0. A processor's local GVT may be out of date, but is always guaranteed to be less than or equal to the true GVT.

[2] There are other techniques (e.g., unit-delay simulation) that exploit this property in pipelined designs by handling clock cycles explicitly in the simulator. Our work is more general in that we allow arbitrary feedback.

Each processor repeatedly executes one of the following operations:

- *Termination.* If GVT is greater than the total simulation time, exit.
- *Update GVT.* Compute the new GVT by initiating a fuzzy barrier and when it succeeds, computing a global minimum.
- *Collect fossil.* Find all regions containing states with times less than GVT and discard such states. It is a local operation.
- *Schedule.* Pick a region, evaluate it and advance its LVT. If the new state generates an event, create the event messages.
- *Send.* Pick the first message from any nonempty fanout message queue. Process the event or anti-message, and rollback or adjust the LVT of the fanout region if necessary. Create the necessary anti-messages.

Because the size and computation requirements for subcircuits vary, the static assignment of subcircuits to processors may not lead to good load balance. However, a dynamic load balancing scheme would incur both *synchronization costs*, for collecting global load information mid-execution, and *communication cost*, due to loss of locality. Statistics from SWEC show that the cost of simulation could double under dynamic load balancing, and our preformance results indicate that load balance is not a major limitation for most circuits.

Memory management in PARSWEC is crucial, because there is no virtual memory on the CM5, and subcircuit states, message queues, and event histories can be large. Subcircuit states are reclaimed by fossil collection. To avoid message buffer overflow, Jefferson proposed a flow control protocol that runs along with Timewarp [4]. Instead, we place restrictions on scheduling and use fixed history sizes. Namely, we require that all event messages be acknowledged, and that a *Schedule* operation for a region r is started only after the previous event messages from r have been processed. We also require that messages be processed in order, so that an anti-message cannot bypass its own event-message. Anti-messages are not stored explicitly, but a count of the events to be canceled is kept. These restrictions make it possible to allocate fixed space for message queues.

To resolve the memory problem for the event histories, we allocate event histories equal to the size of the corresponding state history (which resides on the processor that sends the events). Since an event history is always a subset of its state history, the *Schedule* operation can proceed as long as there is space left *locally*, without flow control for the fanout. Once again, we have traded off space to avoid the communication. The savings in communication for flow control may be significant, since the number of fanouts for an event can be quite large.

Proper scheduling is essential for performance. Regions that will most likely lead to valid states should be evaluated first. In PARSWEC we use the function that predicts the next evaluation time point to define three priority classes in their preferred order: *conservative*, *speculative*, and *unlikely*. An evaluation is *conservative* if the last event times of the fanins are all greater than its evaluation time; it is *speculative* if the next event time estimates of the fanins are all greater than its evaluation time; it is *unlikely* otherwise. Within each class, the regions are scheduled in increasing order of local virtual time; this corresponds to the traditional Timewarp scheduling heuristics.

The frequency of *Update GVT* operations must be balanced between the synchronization overhead of the updates, and the stalls because regions are blocked due to lack of space. *Update GVT* invocation is based on runtime information, since it is not feasible to determine the frequency a priori. The following heuristic works well in practice: a processor starts the first phase of *Update GVT* when a new region runs out of space in its state history, when no region is ready for evaluation, or when the highest priority evaluation is *unlikely*. *Collect fossil* is scheduled immediately after *Update GVT* completes, since that is the only time when states in the histories become fossils.

6 Performance

Speedup is calculated as the ratio of the running time of the sequential SWEC implementation on a Sun/4 (given in Figure 1) to the parallel running time of PARSWEC on the CM5. Most of the numbers were taken on a 64 processor machine, although for a few we had access to a 128 processor CM5. The CM5 nodes contain the same processors as the Sun/4, although the memory systems are different. For most benchmarks, we were unable to run the sequential SWEC on a single CM5 node due to insufficient memory. However, we were able to simulate ADDER on a single CM5 node, and the result showed that the Sun/4 is about 10% faster.

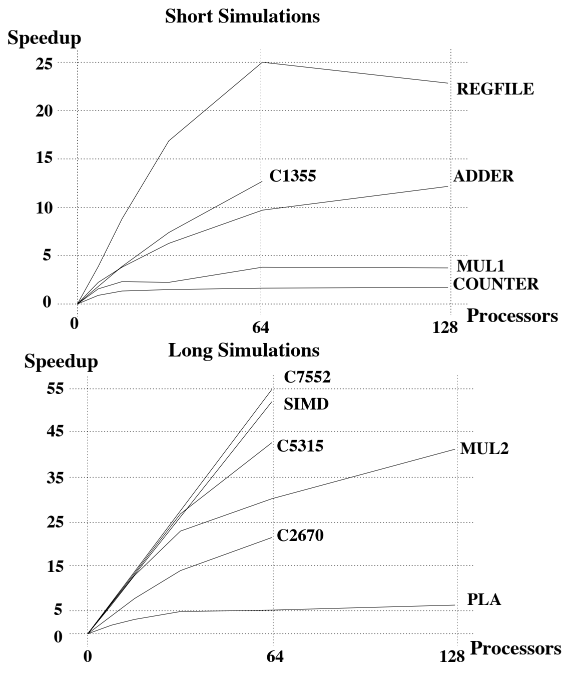

Figure 5: Speedup curves.

The speedup curves are shown in Figure 5. The first graph shows speedups for some of the smaller circuits, all of which took less than 1000 seconds on a Sun/4. Most of these do not have sufficient computation to justify large scale parallelism, but they give the observed parallelism results for the circuits that were simulated on an ideal machine in Section 4. Notice that even with the overhead of optimistic parallelization, the *actual* speedup achieved with PLA is greater than the theoretical speedup achievable by the conservative method.

The peak speedups of our simulator are far greater than those reported for similar timing simulators in [7, 8]. The speedup for SIMD is particularly encouraging, and it shows the feasibility of using large distributed memory multiprocessors to perform large simulations.

The speedup is usually below the theoretical maximum, due to the overhead of data management, scheduling, and communication that arise in practice. Communication overhead is particularly significant; it ranges from 1% of running time for COUNTER and REGFILE, to nearly 17% for PLA and SIMD [3]. Some improvement in performance may be possible by overlapping *Send*s with other computation, although such pipelining tends to complicate programs and have lim-

[3]Despite the communication overhead, we actually observed superlinear speedups for SIMD during the start-up transient. This is because the speculative nature of PARSWEC allows it to relax the strict priority queue in SWEC, which greatly reduces the access costs when the queue is very large.

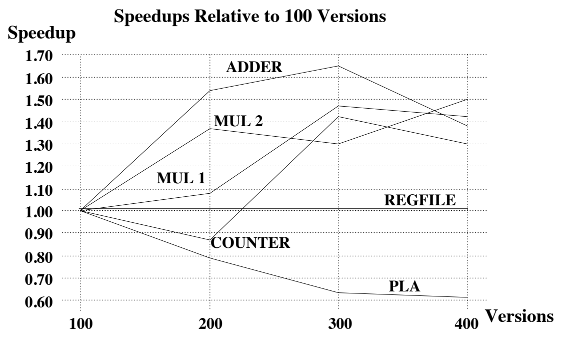

Figure 6: Effect of history size on performance.

ited effect on overall performance.

Another loss of efficiency is due to the static load balancing scheme. The only circuit that showed much effect of this is MUL 1. Post-mortem analysis showed that the activities in MUL 1 are highly concentrated, which is indicated by the imbalance of processor time spent in *Schedule* for each region. Dynamic load balancing might improve the performance of MUL1, but given the added communication overhead, the overall effect is not clear.

The allowed number of states in the histories is a significant tuning parameter in the implementation. All of the simulations in Figure 5 used 100 states per history. To investigate the effect of history size on performance, we also ran some of the benchmarks on 128 processors using history sizes of 100, 200, 300, and 400 states. The results are given in Figure 6. Figure 6 shows that the speedup usually grows with the number of states in the history, with PLA as an exception. The adverse effect of increasing history size for PLA probably comes from the rollback overhead due to excessive speculation, plus the increased cost for managing longer histories.

7 Previous Work

The Timewarp simulation technique has been applied to logic and mixed-level simulation [9], but we are not aware of any Timewarp timing simulator at the level of precision of SWEC. Previous work on parallel timing simulation includes XPSIM[8], an asynchronous algorithm, and CEMU[7], which has both a synchronous and an asynchronous version. CEMU results indicate that the synchronous algorithm is more efficient than the asynchronous one; the synchronous implementation showed speedups up to 20 on a 64-processor hypercube, whereas the asynchronous version showed speedups under 10. Similarly, the speedups achieved by XPSIM were below 4.6 on a 11-processor Sequent. Note that a synchronous implementation uses a uniform time step algorithm, which requires more total computation than the variable time step algorithms of XPSIM and SWEC. In general, variable time step algorithms have better absolute performance.

8 Conclusions

In this paper we showed that the amount of parallelism in timing simulation is sufficient to justify the use of a large multiprocessor, and that optimism is essential for exploiting parallelism in sequential circuits. The speeup for the PLA circuit is higher than the theoretical maximum for conservative simulation, and the best speedups, 50 on 64 processors, are better than those obtained in previous efforts. We are currently exploring the use of lower level parallelism within subcircuit evaluation, and looking at other simulation domains in which optimistic scheduling might be beneficial. Because the PARSWEC implementation involves nontrivial protocols, we hope to leverage off the current implementation by providing compiler and runtime support that will be useful across problem domains.

Acknowledgements: We would like to thank Shen Lin, Bertrand Irissou, John Wawrzynek, and Jeff Jones for their comments and help.

References

[1] M. Marek-Sadowska S. Lin, E. Kuh. Swec: A stepwise equivalent conductance simulator for cmos vlsi circuits. In *Proc. of European Design Automation conference*, February 1991.

[2] R. Saleh et al. Parallel circuit simulation on supercomputers. *Proc. of the IEEE*, 77(2), December 1989.

[3] L. Snyder M. Bailey. An empirical study of on-chip parallelism. In *Proc. of 25th Design Automation Conference*, 1988.

[4] D.R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3), July 1985.

[5] J. Misra K.M. Chandy. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11), April 1981.

[6] J. Pal Singh et al. Splash: Stanford parallel application for shared memory. *Computer Architecture News*, 20(1), March 1992.

[7] B. Ackland E. DeBenedicts. Circuit simulation on a hypercube. In *Distributed Simulation Conference*, 1988.

[8] T. Lee. Parallel circuit simulation: A case study in parallelizing programs. Technical Report Master thesis, Computer Science Division, University of California, Berkeley, 1991.

[9] Jr. et al. J.V. Briner. Breaking the barrier of parallel simulation of digital systems. In *Proc. Design Automation Conference*, 1991.