PARALLEL SWITCH-LEVEL SIMULATION of VLSI CIRCUITS

Yu-an Chen, Vikas Jha, and Rajive Bagrodia
Computer Science Department,
University of California, Los Angeles, CA 90024

Computer Science Tech Report:950020, June 1995, UCLA

Abstract

This paper presents the results of an experimental study to evaluate the effectiveness of parallel simulation in reducing the execution time of switch-level models of VLSI circuits. Specific contributions of this paper include: (i)Parallelization of an existing switch-level simulator such that it can be executed with a variety of simulation protocols (almost all previous parallel circuit simulators have focused on gate-level models and have targeted either conservative or optimistic simulators, but not both). (ii) Adapting existing circuit partitioning algorithms designed for gate-level models to switch-level designs and evaluating their effectiveness for parallel simulation. (iii) Demonstrating speedups with both conservative and optimistic simulation protocols for a set of combinational circuits (almost all previous studies on parallel simulation for VLSI circuits have failed to extract speedups with conservative protocols); in particular our experiments showed that it was possible to obtain a speedup of upto 4.2 for conservative algorithms, and upto 5.8 for optimistic algorithms among five circuits, ranging in size from 3K transistors to about 68K transistors.

1 Introduction

VLSI circuit designs may be simulated at many levels including the more abstract functional levels or the more detailed gate, switch and circuit levels. In general, the simulation at a more detailed level is both more accurate and more expensive than simulating the abstract design. Due to the resource-intensive nature of VLSI design simulation, designers often use high-level or behavioral simulations where the circuit is simulated at a functional level by modeling it as a synchronous data-flow graph[19]. Clearly, this approach is not an effective substitute for simulations at the gate and switch-level as higher level simulations by their very nature preclude detailed logic and timing verification.

A number of hardware solutions have also been used to increase the execution efficiency of simulation models, including the use of special-purpose hardware such as the Yorktown Simulation Engine (YSE)[22] or FPGA-based circuit emulation. Unfortunately the hardware tools are generally expensive, and have limited applicability.

This paper explores the use of general purpose parallel machines for execution of VLSI models. Parallel circuit simulation can be either synchronous or asynchronous. As discussed in the next section, synchronous parallel simulation does not appear to offer significant potential for improvement with parallel execution. For asynchronous simulations, the circuit is first *partitioned* such that each portion can potentially execute on a separate processor, and a *synchronization mechanism* is used to ensure that incoming signals at each partition are processed in their correct global order. The two primary synchronization mechanisms that have been used include the conservative parallel discrete-event simulation (PDES) protocols[21] and the optimistic PDES protocols[16].

This paper presents experimental results on the simulation of *switch-level* models of VLSI circuits using both conservative and optimistic synchronization protocols.

Primary contributions of this paper include the following: *first*, to the best of our knowledge, this is the first study that evaluates the speedup potential for switch-level models with multiple parallel simulation protocols. Almost all preceding studies have concentrated on gate-level models[25, 27, 9]. Switch-level models are more time-consuming than gate-level models and exhaustive use of switch-level models can uncover timing errors in the design that may not be discovered with gate-level models. *Second*, this is the first study that directly compares the speedup potential for conservative and optimistic simulation techniques for switch-level models. An existing switch-level simulator called IRSIM was implemented in an algorithm-independent simulation language called Maisie. The resulting program, called MIRSIM, may be used to input a partitioned circuit description and simulate the corresponding circuit using a set of parallel simulation protocols. Third, the results from this paper show that the communication topology generated by a partition is more important than the size of the cut-sets in determining the performance of both conservative and optimistic execution of switch-level models. Lastly, the paper demonstrates that given appropriate circuit partitionings, both conservative and optimistic techniques can yield speedups in parallel circuit simulations. Previous studies have invariably concluded that conservative techniques are not viable for parallel circuit simulation.

The next section describes related work in the area of parallel circuit simulation. Section 3 describes the IRSIM simulator, and its parallel implementations, including the partitioning algorithms used in this study. Section 4 describes the benchmark circuits used in the experiments and presents the results of both sequential and parallel simulation of these circuits. Section 5 is the conclusion.

2 Related Work

Prior work in parallelizing circuit simulations have used both synchronous and asynchronous methods. Synchronous parallelism exploits concurrency if the circuit has a large number of simultaneous events. However previous measurements by Wong et al[29] reported that in small circuits ranging from 347 to 3,680 elements, the average activity level was only about 1.2% and the average concurrency level (i.e. average number of simultaneous events) of between 18 to 30. Bailey et al[8] showed that contrary to expectations, this concurrency level does not scale linearly with circuit size; thus although the absolute concurrency levels were found to increase with circuit size, the relative concurrency measured as a percentage of the size of the circuit actually decreased for larger circuits. It is generally accepted that synchronous parallelism is unlikely to yield significant and scalable speedups[25].

Previous work on the application of asynchronous techniques has been done primarily for logic simulation using gate-level models, rather than switch-level models. This includes the work by Su & Seitz[27], who studied the effectiveness of six variants of the conservative simulation protocol based on deadlock-avoidance using null messages using a second generation iPSC/2 Hypercube. Recently, Soule [24] applied the conservative techniques to the logic level simulation of a number of benchmark circuits ranging from a small 5,000 element 16-bit multiplier to a large 24,600 element multiprocessor directory cache controller. The models were executed on a 16-processor Encore Multimax. The performance of the parallel models was also estimated by simulating their execution on a detailed simulation model of the Stanford DASH multiprocessor. The researchers obtained only slight speedups for the various circuits and noted that the deadlock resolution overheads limited the speedups.

Optimistic techniques have also been used for gate-level simulations. The most successful parallelization have been obtained by Briner et al[9] who evaluated the speedups for logic simulation of circuits using a window-based optimistic protocol.

Speedups were reported for circuits ranging from a small adder that had 3360 transistors, to a CC array with 31,680 large multi-input gates, and a QRS chip with approximately 45,000 transistors. Reported speedups measured on 32 nodes of a BBN Butterfly ranged from 7 for the adder to almost 25 for the CC Array. Good speedups have also been reported by Sporrer et al[26] using TimeWarp on a network of workstations for logic simulation of the ISCAS89 benchmarks. The researchers used a hierarchical, cluster-based partitioning algorithm and reported a peak speedup of about 8 using 20 workstations; however the speedups were reported relative to a 1 processor implementation of the distributed simulator, rather than with respect to a sequential implementation. Finally, there has been some recent work in the use of optimistic techniques for parallel simulation of VHDL models of VLSI circuits[20].

3 Switch-level simulator

IRSIM[28] is an event-driven, switch level simulator that incorporates a simple linear model of MOS transistors. In IRSIM, MOS transistors are modeled as voltage controlled switches in series with fixed resistances, while gates and interconnects are modeled as fixed capacitances. Standard RC network techniques are used to compute the final logic state of each node, as well as the transition delay.

The electrical network in IRSIM consists of two kinds of data structures, *wires* and *transistors*, where the *wire* represents the connection point between two transistors. For each *wire* and *transistor*, there are three logic states, *High, Unknown, Low*, associated with them. In general, the state of a *transistor* is changed if the state of a *wire* which is connected to the gate of the transistor is changed. After that, the state of *wires* which are connected to source and drain of the transistor are recalculated. If the *wires* change state, all other *wires* which are affected by that change are examined. For each affected *wire* that will change state, the value of the new state and the expected time when the change will happen is computed. An event corresponding to that change is generated and added in the time order to the global event list in the simulator. The IRSIM simulates repeatedly executes events from the global event list until there is no event in it or reaches the end of simulation time.

3.1 Parallel Implementation

A parallel implementation of IRSIM, called MIRSIM, was designed by implementing the IRSIM simulator in Maisie[4]. Maisie is among the few existing languages that supports the execution of a discrete-event simulation model with multiple algorithms and provides constructs to reduce the simulation overheads with both conservative and optimistic parallel algorithms. A program written in Maisie is independent of any synchronization algorithm. When it is compiled, the analyst can indicate the specific simulation algorithm that is to be used to synchronize execution of the model: sequential, parallel conservative, or parallel optimistic. The compiler generates the appropriate code to interface the model with the corresponding run-time system: a splay-tree based implementation of the global event-list algorithm for the sequential implementation, a null-message or conditional event implementation of the parallel conservative synchronization algorithm, or a space-time implementation of the optimistic synchronization algorithm. The compilation of the Maisie program to optimize performance with respect to a specific synchronization algorithm is beyond the scope of this paper.

A Maisie program is a collection of entity definitions and C functions. An entity definition (or an entity type) describes a class of objects. An entity instance, henceforth referred to simply as an entity, represents a specific object in the physical system and may be created and destroyed dynamically on local and remote processors. Entities communicate with each other using buffered message-passing. Maisie defines a type called **message**, which is used to define the types of messages that may be received by an entity. Every entity is associated with a unique message-buffer. A message is deposited in the message buffer of an entity by executing a Maisie send statement. An entity accepts messages from its message-buffer by executing a blocking receive statement. The statement can specify an optional timeout interval, such that if a message of the type specified by the entity is not received by it within the timeout interval (in simulation time), the entity is sent a timeout message. Thus the wait statement can also be used to schedule simulation events. Every Maisie program must contain an entity called *driver*. When execution of a Maisie program begins, an instance of this entity is created automatically by the run-time system on one processor to initiate the simulation. Subsequently, user-defined entities are spawned among the available processors as specified in the body of the *driver* entity.

The Maisie simulation environment has been implemented on a variety of workstations, on networks of workstations and on distributed memory multicomputer like the IBM SP1. Maisie implementations have yielded good speedups for queuing network benchmarks with both conservative and optimistic algorithms, with speedups approaching close to linear for large grain computations[18, 2] The IBM SP1 implementations have been used for gate level circuit simulations[6, 5] and for simulation of parallel programs[23].

The MIRSIM program contains a *driver* entity and one other entity type called *irsim*. The *driver* entity initiates the simulation by instantiating a set of *irsim* entities. The circuit to be simulated is partitioned into a number of subcircuits and each *irsim* entity is used to simulate a unique partition. The partitioning algorithms used in this paper are described in the next section. The *driver* entity also reads the command file, which contains the input vectors and the commands that specify the operations to be executed on the circuit, and sends the relevant information to each *irsim* entity.

Each *irsim* entity reads in the subcircuit corresponding to its partition. Subsequently it simulates the execution of its partition using the same model as the original IRSIM simulator. However, the data structures and program control flow had to be modified significantly. We briefly discuss the primary modifications, first to the data structures and subsequently to the control logic. The IRSIM simulator was written specifically for sequential implementation and relied heavily on global data structures. As Maisie entities communicate only via messages, (to facilitate their distribution among multiple processors), it was first necessary to transform the program such that global data structures were replaced by data structures declared locally inside an entity, and all read and write access to the global data was implemented via message communication among the corresponding entities. Another major modification was in the design of local data structures for the *irsim* entity. As discussed subsequently, optimistic execution of an *irsim* entity requires that its state be checkpointed periodically. The size of an *irsim* entity's state is directly proportional to the size of the circuit being simulated. As circuits with 50,000 transistors are not unusual, the checkpointing costs may become prohibitive. To reduce this cost, the state of an *irsim* entity was subdivided into two parts: *static-value* data structures, whose value are initialized once and remained unchanged during the simulation, and *dynamic-value* data structures whose values were changed dynamically. *Static-value* data structures are tagged to reduce the size of the state to be checkpointed for an entity.

With respect to the control flow, in the sequential implementation, the entire circuit was simulated as a single object; signal flow among transistors was communicated using a single global event-list. However, in a parallel implementation on a distributed memory architecture, the circuit is partitioned and the global event list is also split among partitions. Moreover, those 'wire' data structures which connect transistors in different partitions are duplicated in subcircuits in order to keep the connection information. In order to maintain the consistency between these duplicated wires, messages must be exchanged between partitions if one of duplicated wires changes it status. The clustering and partitioning methods used in our experiments attempt to minimize the number of messages that are exchanged among the partition entities.

3.2 Circuit Clustering and Partitioning

To support parallel execution, the circuit being simulated must be divided into a number of partitions, such that a partition can potentially be simulated on a separate processor. In this paper, we used two different partitioning algorithms: the first algorithm, referred to as K-FM is the well-known K-way generalization of the 2-way iterative partitioning algorithm proposed by Fidducias-Mattheyeses[15] and the second algorithm, called K-AFM, is used to generate *acyclic* partitions[13]. Most existing partitioning algorithms, including the K-FM algorithm, formulate the problem as follows: given a circuit containing N connected components, decompose the circuit into K disjoint but complete partitions. The circuit is modeled as a graph $G(C, N)$, where C is a set of components and N is a set of nets. A component may has multiple input ports and only one output port. A net connects one output port of a component to input ports of multiple components. A net is cut if the components connected by the net are put in the different partitions. The objective function used by most algorithms is to minimize the number of net cuts between the partitions, while ensuring that each partition has an approximately equal area. The area may be represented either by the number of components, nets, or a combination of the two metrics.

Rather than partition the original transistor-level circuit directly, it is preferable to first use a clustering technique to ensure that certain transistors will belong to a common partition. The IRSIM model favors the use of the following clusterings[1]: first, all transistors that belong to a chain must be stored in the same partition. The chain for a transistor refers to the set of transistors that are connected to the given transistor through their source or their drain. It is preferable to have the chain in one partition because when a transistor changes its state, its output voltage and state transition time is affected by the status of other transistors in its *chain*, Second, all transistors with gates that are connected together must receive a signal at the same time and are perhaps handled most efficiently by storing them in a common partition. For those transistors with two properties mentioned above are put in the same cluster. The set of components C of the graph G in the partitioning algorithm is mapped from the set of clusters of transistors in the circuit. A net is existed among components if there are connections from the sources or drains of transistors in one cluster to the gates of transistors in the other clusters.

Use of the preceding clustering technique normally results in clusters that may contain a widely differing number of transistors. To facilitate the generation of balanced partitions, the clustering phase maintains two metrics with each cluster: the number of transistors and the total number of output signals; the former represents the area of the cluster and is used in the generation of balanced partitions, and the latter is used to determine the cut-size of partitions for that cluster.

The clustered circuit is partitioned using the K-FM and K-AFM algorithms. The K-FM algorithm starts with a randomly generated, balanced, initial partition and subsequently moves components(or clusters) among the partitions. At each step, feasible components are first identified. A component is feasible for moving, if moving it to another partition will not violate the balance constraint. Among all feasible components in a partition, the component with the highest gain is selected, where the *gain* of a component is defined to be the amount of reduction in the size of the cut-set that results from moving the component to the other partition. When all feasible moves have been inspected, the best partition encountered during the current iteration is saved as the initial partition for the next pass. The algorithm terminates either after a specified number of iterations have been executed or when an iteration makes no improvement to the partitioning solution.

A parallel simulation contains two primary sources of overhead: message communications and synchronization. The former can be reduced by minimizing the number of messages. Each wire in the original circuit that is cut by the partitioning algorithm potentially adds a message to the simulation because a change in the signal on the corresponding wire must be communicated to the other partitions. Thus, in the preceding algorithm, the component moving heuristic attempts to reduce the size of the cut-set to reduce this overhead. Synchronization overheads depend on a number of factors as discussed in the next section. However, other things remaining the same, acyclic topologies can typically be simulated more efficiently than cyclic topologies. To test the applicability of this idea for circuit simulations, the second algorithm considered in this paper used acyclic circuit partitionings as described next.

We say that a directed edge exists from partition P_i to P_j if there exists a wire in the original circuit which the signal always propagates from the transistors in partition P_i to P_j. For an acyclic partitioning, it must be the case that given any two partitions, say P_i and P_j, if a path exists in the partitioning from P_i to P_j, no such path exists from P_j to P_i. Given a circuit that does not contain *any* feedback, it is possible to modify the preceding algorithm to yield acyclic partitions; the modified algorithm is referred to as the K-AFM algorithm. For this purpose, the circuit representation is required to include signal direction along with the connectivity information. With this additional information, the preceding K-FM partitioning algorithm can be used to generate acyclic partitionings by imposing additional constraints that maintain an acyclic dependency graph. This extension, referred to as K-AFM uses an initial acyclic partitioning based on a random topological ordering of the initial circuit. The criteria for choosing a feasible move in the K-AFM algorithm is modified to select a component that maximizes gain under both acyclic and area constraints. As this algorithm imposes an additional constraint on the partitioning process, the partitionings generated by it typically yield partitions with a larger cut size than the standard K-FM algorithm described earlier.

The clustering step described earlier can introduce cycles among the clusters. To facilitate its use with acyclic partitioning algorithms, we attempt to remove cycles in the clustered circuit by collapsing them into a super cluster as long as the area constraint is not violated. Although this is not feasible in general, for the circuits simulated in this paper, it was always possible to do so. A general technique to partition a circuit with feedbacks such that there are no cyclic logic dependencies among subcircuits in different partitions *within the same clock cycle* is described in [14].

A complete description of the K-AFM algorithm and its performance with respect to the size of the cut-set relative to the K-FM algorithm is in [13]. This paper explores the impact of the clustering and partitioning algorithms on the parallel simulation of *switch-level models.*

3.3 Synchronization

A partitioned circuit can be simulated on one or more nodes of a parallel architecture. Execution on multiple nodes must be synchronized to ensure that events are simulated in their correct global order. Two primary synchronization mechanisms have been used in the literature: conservative and optimistic. Conservative algorithms do not permit any causality error; a simulation object (also called an entity) cannot process a message until the system can guarantee that it will not subsequently receive a message with an earlier timestamp. This constraint may introduce deadlocks, which are typically handled by incorporating deadlock detection[21] or deadlock avoidance[21, 10, 11] mechanism into the simulation algorithm. One of the most commonly used such mechanism is *null* messages. A null message refers to a message sent by one LP to another to indicate a lower bound on the timestamp of any future messages that may be sent by the entity. If a null message with timestamp T is sent at simulation time t, $T > t$, then (T-t) is referred to as the lookahead of the entity.

The transmission and processing of null messages is the primary source of overhead in communication schemes. In general, the performance of conservative schemes is heavily influenced by the lookahead properties of the entities, as a high lookahead typically implies that fewer null messages need to be sent. In addition to good lookahead, if the entities have a sparse communication topology with a predictable communication pattern, the conservative schemes perform very well as relatively few null messages need to be sent. In general, for a network with a cyclic topology and poor lookahead, null messages can quickly overwhelm the simulator leading to poor parallel performance. Conservative implementations of the Maisie simulation language are described in [7].

Optimistic algorithms[17, 12] allow an object to process messages out of order; causality errors are corrected by using rollbacks and recomputation. Implementations of optimistic algorithms are usually more difficult because they require complex mechanisms for detection and handling of causality errors, termination detection, exception handling, and memory management. In order to correct causality errors, the state of each entity must be checkpointed periodically. If an entity receives a straggler message, i.e. a message with a timestamp smaller than its current simulation time, it is rolled back to the state with simulation time before or equal to the timestamp of the straggler message. The number of events that are canceled during a rollback is referred to as the rollback distance. When an entity is rolled back, all messages sent by the entity during computations that are rolled back must also be canceled. The entity begins to execute again from some checkpointed state prior to the timestamp of the straggler message, processing the straggler message in its correct timestamp order. A checkpointed state with timestamp t is 'garbage collected', when the Global Virtual Time (or GVT) of the simulation exceeds t. The GVT is defined as the minimum timestamp of any message (or event) in the system. Optimistic implementations incur a number of overheads including checkpointing, rollback, and GVT computation, which must be reduced to derive reasonable benefit from parallel execution of the simulation model. Details of the optimistic implementation of Maisie designed to reduce many of these overheads are described in [4, 3].

4 Experiments and Results

Five different circuits were used to measure the performance of MIRSIM. The largest, a 64-bit squaring circuit called $Square64$, consists of approximately 68K transistors. The smallest circuit, an 8x8x8 dynamic RAM called $Chip2u$, has about 3K transistors. The other three circuits were $Square$, a 32-bit squaring circuit, $DDFS$, a Direct Digital Frequency Synthesizer and $Filter$, a DSP signal filter. Figure 1 shows the number of transistors included in each circuit.

Circuit	Square64	Square	Filter	Chip2u	DDFS
# of Trans.	67972	15620	13464	3336	17077

Figure 1: Circuits simulated by MIRSIM.

The sequential and parallel measurements reported in this paper were executed on one or more nodes of an IBM 9076 SP1 parallel computer. The IBM SP1 is a distributed memory machine that consists of a set of RS/6000 workstation processors connected by a high speed switch. Each node has a main memory of 128 megabytes and is connected to a 1 Gigabyte disk that is used for temporary file storage.

In Figure 2, we compare the performance of IRSIM simulator with that of a uniprocessor implementation of MIRSIM, using the five benchmark circuits. As seen in the table, the two simulators have almost identical performance, with the uniprocessor performance of MIRSIM at most 5-10% worse than that of IRSIM. In any case, all of the speedup measurements that have been reported in this paper have been computed with respect to the time taken by the original IRSIM simulator running on one node of the IBM SP1.

Circuit	Square64	Square	Filter	Chip2u	DDFS
IRSIM	114.8	15.1	55.7	43.7	21.1
MIRSIM	120.0	16.5	58.6	49.3	22.9

Figure 2: Execution time(sec) of IRSIM Vs Uniprocessor MIRSIM

Next, we describe the results from the partitionings used in our experiments. The circuits were first clustered, and then partitioned into 2, 4, 6, 8, and 16 partitions using the two algorithms described in the previous section. Figure 3 lists the size of the cut-sets obtained for each partition. As seen in the table, the K-FM algorithm yields partitions that have much smaller cut-sets as compared with the acyclic K-AFM algorithm, for most of the circuits. However, as as seen later in the section, the performance of the parallel simulator with the K-AFM algorithm was considerably superior.

Partitions		2	4	6	8	16
Square64	K-FM	91	240	366	513	990
	K-AFM	1289	2548	3049	2954	4018
Square	K-FM	36	96	147	192	363
	K-AFM	311	620	724	654	955
Filter	K-FM	91	191	248	298	435
	K-AFM	183	343	391	509	715
Chip2u	K-FM	63	96	111	119	127
	K-AFM	66	91	108	114	122

Figure 3: Cut-sizes of K-FM and K-AFM partitions.

Next, we describe the results of parallel MIRSIM experiments using optimistic and conservative methods.

4.1 Optimistic Synchronization

In an optimistic protocol, events may be processed in an incorrect order. These violations must be detected and corrected. Therefore, the state of an entity is checkpointed periodically. The checkpointing(state saving) frequency has been known to have a significant impact on the performance of optimistic simulators. Therefore, we first ran a set of experiments to empirically identify the optimal state saving frequency. Let S refer to the interval with which the state of an entity must be checkpointed, i.e. the state of an entity is checkpointed after it has executed S events.

The total time spent by the entity in checkpointing its state, therefore, decreases as S is increased. However, as S is increased, the average rollback distance i.e., the average number of messages that must be canceled and possibly re-executed when the entity rolls back, increases. This may cause the time spent by an entity in processing rollbacks to increase causing the total execution time for the entity to increase. Figure 4 and Figure 5 plot the state saving time and rollback time as a function of the state saving interval time for the *square* circuit. As seen from the graphs, as S is increased, the total execution time for the program first decreases, but eventually begins to increase, as expected. The value of S which corresponds to the minimum execution time for the simulation was chosen as the state saving interval for all experiments reported in this paper. From the graphs, we can see that the optimal value of S is 26. All subsequent experiments reported in this paper used a value of $S = 26$ for checkpointing every entity.

We next consider the speedup obtained for each circuit using the optimistic runs. In each case, speedup is defined as T_{seq}/T_{par}, where T_{seq} measures the execution time for the circuit using IRSIM, and T_{par} is the execution time for the parallel implementation of MIRSIM.

Figure 6 and Figure 7 present the speedup obtained for circuits using 2,4,6,8, and 16 processors, for the two partitioning methods. For a n-processor implementation,

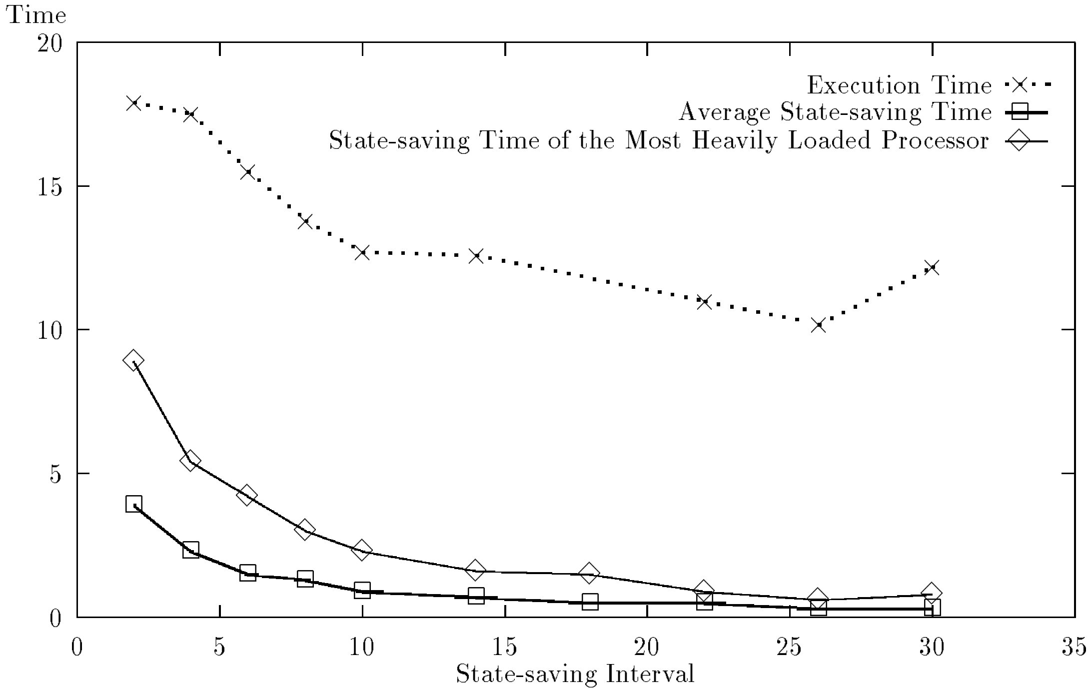

Figure 4: State saving overhead vs. State-saving interval (Square)

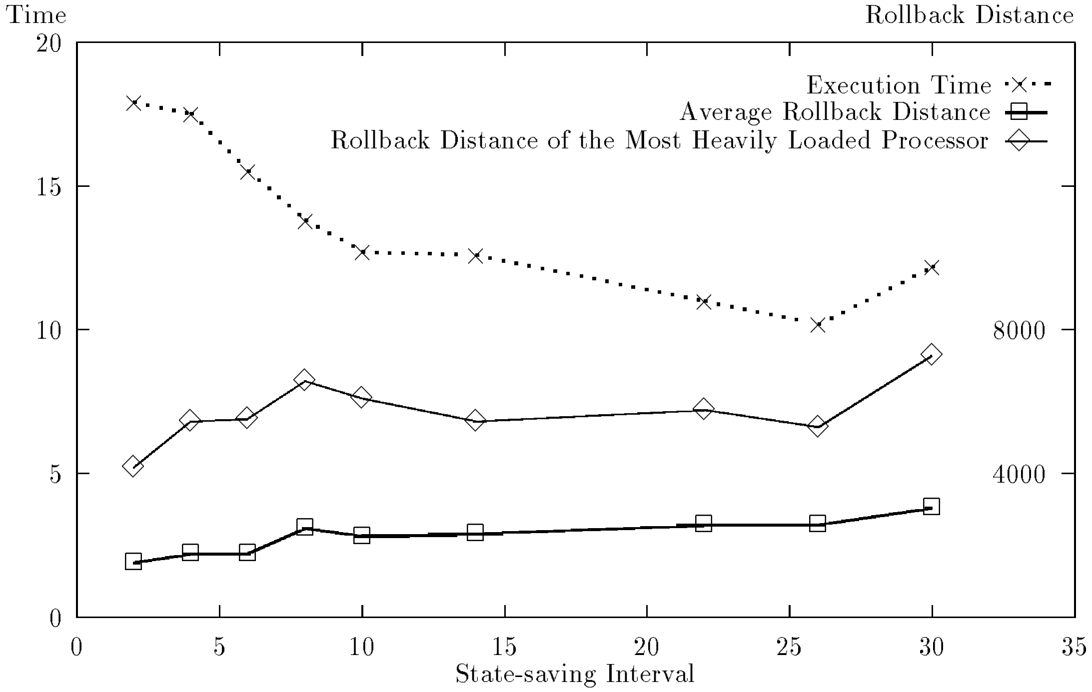

Figure 5: Rollback Distance vs. State-saving interval (Square)

the circuit was partitioned into n components such that each processor always contained one partition entity. As DDFS is a sequential circuit, it was not partitioned using either the K-FM or K-AFM partitioning algorithms as the K-AFM method is not guaranteed to produce acyclic partitions for sequential circuits. Instead, we use a user-specified partitioning to partition the circuit on the basis of its high level layout, with the aim of minimizing the number of cycles and the message communications among the partitions. (note that, for simplicity, the speedup obtained for DDFS using this partitioning scheme is shown in the K-AFM graph). As seen from these figures, the K-AFM partition always performs better than the K-FM partition. This is primarily because acyclic partitions produce fewer rollbacks, which is the main source of overhead for optimistic executions. We next analyze the main sources of overheads for the optimistic executions.

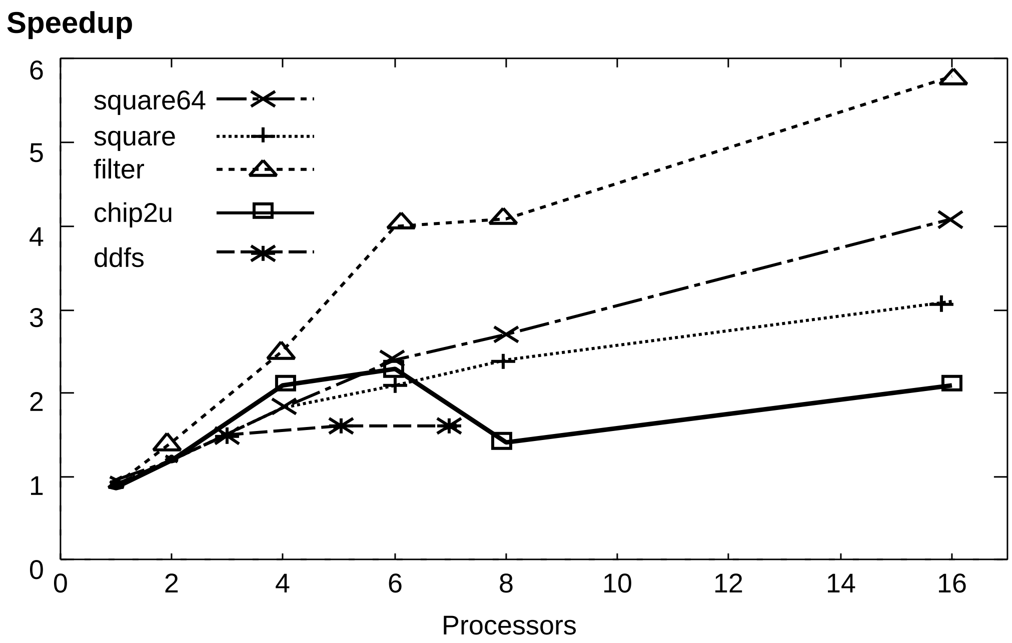

Figure 6: Optimistic Speedups using K-AFM partitioning.

Figures 8 to 11 depict measurements that reflect the cost of various components of an optimistic simulator. For each circuit, the amount presented in the graph is the average of the values computed across all entities in the simulation. The first

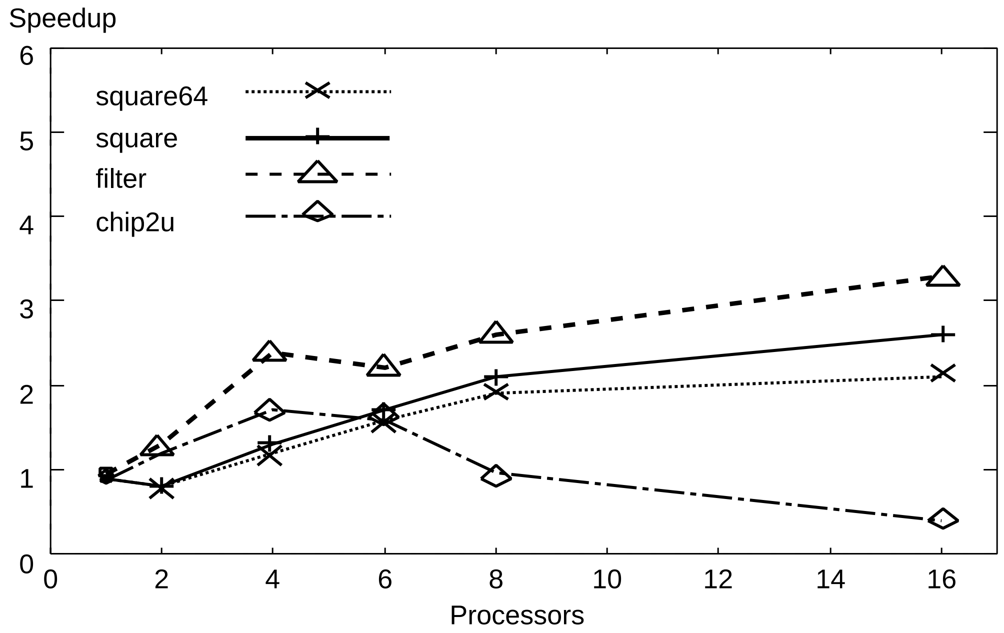

Figure 7: Optimistic Speedups using K-FM partitioning.

column in each graph, titled *entity computation* represents the percentage of time spent in processing events. This includes the time spent in processing incorrect events (i.e. events that are subsequently rolled back). The second column, titled *correct computation* shows the portion of the computation that is spent in computation that is not subsequently rolled back. *State saving* refers to the average percentage of total time spent by an entity in checkpointing the state of the computation, and *blocking* refers to the average percentage of time that each entity was idle because it had no messages to process. The last column in each graph titled *Canceled Message* measures the average percentage of messages sent by an entity that were subsequently canceled.

A comparison of the costs in the first two columns for the K-FM and K-AFM partitioning algorithms shows that although the K-FM partition results in better utilization of the machine, a significant portion of this time is spent doing incorrect work, i.e. computation that is eventually canceled. This conclusion is also supported by looking at the *Canceled Message* column, which shows that a much smaller percentage of total messages sent in the K-AFM partition were canceled as compared with the K-FM partition. Measurements for individual entities showed that only 0 to 2% of the messages were canceled for K-AFM partition, but 8 to 28 % of messages were canceled for the K-FM partition.

Thus, a partitioning like the FM which has less blocking is not necessarily better, because a significant percentage of the execution time may be spent on incorrect computation. The high percentage of correct computation and a high blocking time in the K-AFM partition indicates that for the most part, messages arrive at an entity in more or less increasing timestamp order; however, messages arrive in waves with a significant time elapsing between successive waves.

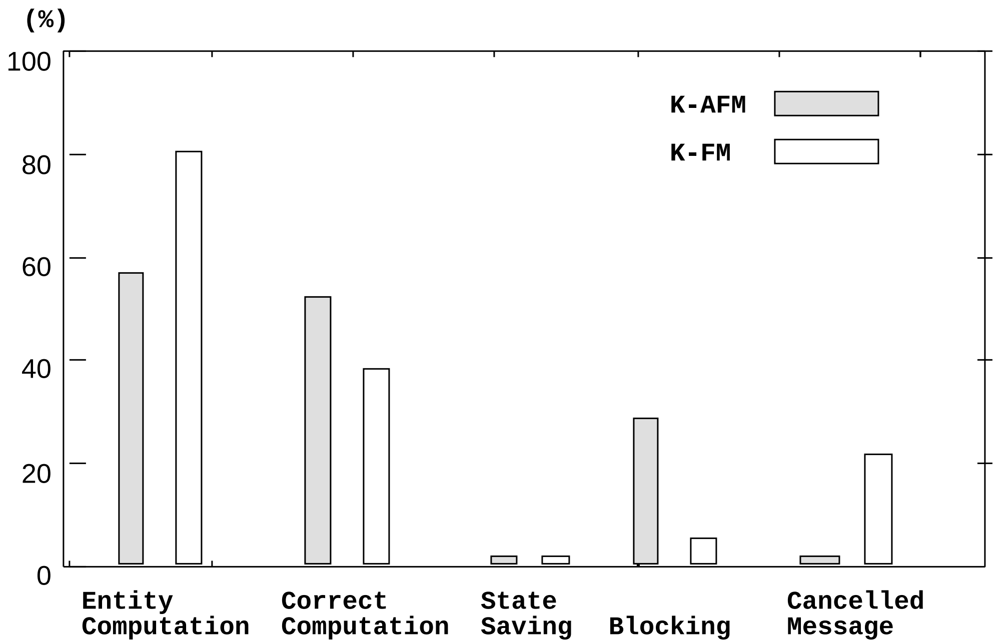

Figure 8: Square64 circuit.

The state saving overhead was quite small for all the circuits. This is because Maisie allows the user to specify two types of data structures : a read-only part that is initialized in the beginning and doesn't change during the course of simulation, and a read-write part that may change during the simulation. Clearly, the former

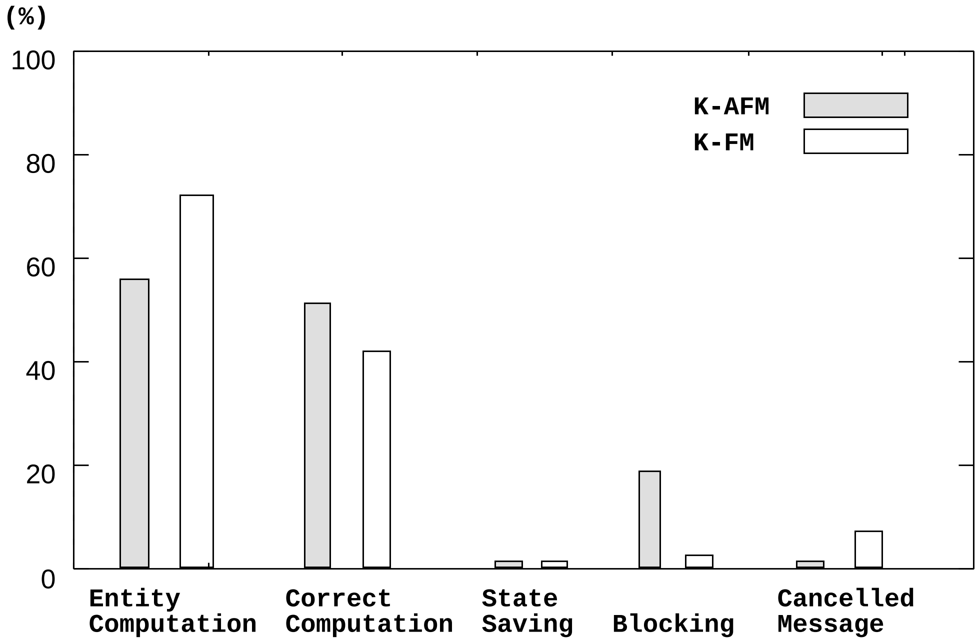

Figure 9: Square circuit.

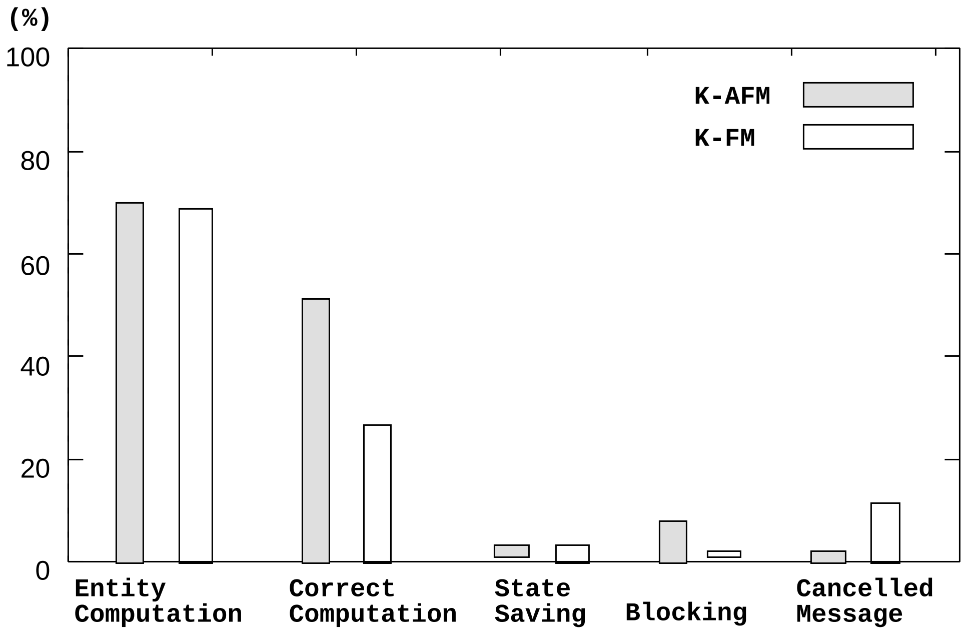

Figure 10: Filter circuit.

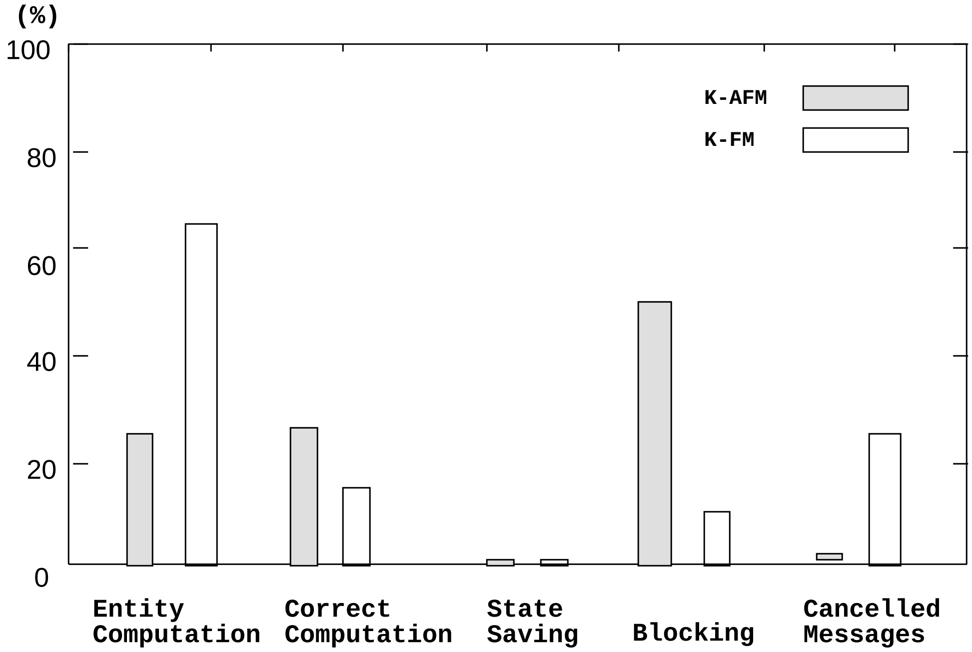

Figure 11: Chip2u circuit.

does not need to be saved every time that the state of the entity is checkpointed. In MIRSIM, most of the data structure (state) is of the read-only type that does not need to be checkpointed, leading to a low state saving overhead. For example, even for a large circuit like $Square64$ (which has 68K transistors), only 1.05% of the total execution time is spent in state saving overheads.

4.2 Conservative Experiments

Maisie has been implemented using 3 different conservative protocols, including the null message protocol[21], conditional event protocol[11], and an asynchronous combination of the null message and conditional event protocols. The implementation of these protocols is described in detail in [18]. A MIRSIM program can be transparently synchronized using any one of the three protocols. However, the experiments reported in this paper were carried out using the (eager) null message protocol, as this was found, empirically, to be the most efficient protocol for switch-level models.

As discussed in the previous section, the performance of conservative algorithms is profoundly affected by the static message communication topology of the application. In particular, other things remaining the same, the presence of cyclic topologies can significantly degrade its performance. Our experience with switch-level simulators confirmed this hypothesis. The K-FM method generally yielded poor speedup because of the presence of numerous cycles and a higher connectivity. The null message overhead for K-FM partitions was found to be 4 to 5 times more than that for the K-AFM partitions. As a result, we only report the performance of the conservative implementation with the K-AFM partitioning method (and the user specified partitioning for the $DDFS$ circuit).

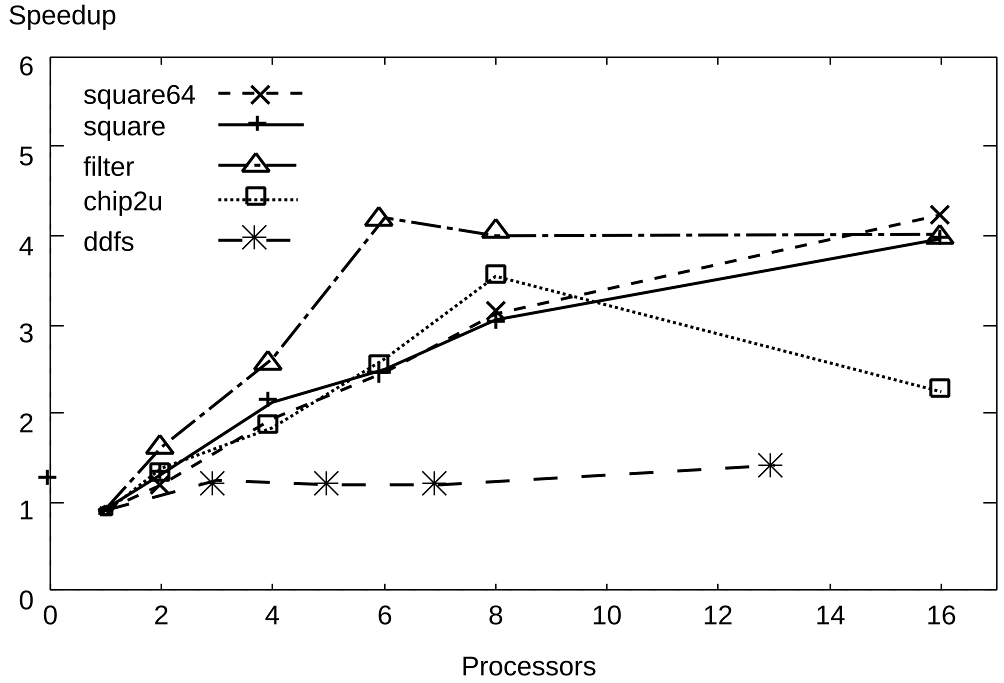

Figure 12: Conservative speedups using K-AFM partitioning.

Figure 12 plots the speedup (with respect to the execution time for IRSIM) for the five circuits. The four combinational circuits perform quite well yielding speedups ranging from 3.55 to 4.23, whereas the sequential circuit, DDFS, only yields a speedup of 1.42. This is because, for the combinational circuits, K-AFM produces relatively balanced acyclic partitions. Simulation of successive test vectors is easily pipelined for such a partition, resulting in a higher level of parallelism. However, the user defined partition of the DDFS circuit results in an unbalanced partition (most of the computation is concentrated in 3 partitions) because of an attempt to keep the partition acyclic. Figure 13 shows the Null Message Ratio (NMR) for each circuit. NMR is defined as the ratio of the total number of null messages sent in the simulation to the total number of regular messages sent. As seen from the table, NMR is quite low for all the circuits including DDFS. This confirms the hypothesis that the poor speedup for DDFS is due to poor load balance, and not because of a high null message overhead.

Circuits	Square64	Square	Filter	Chip2u	DDFS
Null Message Ratio (NMR)	0.10	0.23	0.73	0.04	0.15

Figure 13: Null Message Ratio for K-AFM partitions.

5 Conclusion and Future Research

We have shown the feasibility of applying parallel simulation techniques to the problem of switch level VLSI circuit simulation. A parallel implementation of IRSIM, a switch-level simulator, was designed by implementing IRSIM in Maisie, a C-based parallel simulation language. The parallel simulator, called MIRSIM was executed using both conservative and optimistic parallel simulation protocols. Good speedups were obtained for four of the five circuits (ranging in size from 3K transistors to about 68K transistors) with both optimistic and conservative methods.

Figure 14 summarizes the best speedups obtained using the conservative and op-

Circuit	Square64	Square	Filter	Chip2u	DDFS
Conservative	4.2	4.0	4.2	3.6	1.4
Optimistic	4.1	3.1	5.8	2.5	1.5

Figure 14: Best speedups of MIRSIM (wrt IRSIM)

timistic protocols. Both the protocols yield relatively good speedups for the combinational circuits. The main reason for this is balanced partitions and acyclic topology. Acyclicity reduces the rollbacks for the optimistic protocols and allows better pipelining (of the test vectors) for the conservative protocol. The conservative performance of the Filter circuit was worse than the optimistic performance. This could be attributed to the fact that some of the channels in the circuit were much more active than others (as revealed by a profile of the message communication pattern). This hurts the performance of null message protocol more than that of the optimistic protocol. The sequential circuit DDFS performed poorly with both the protocols because of poor load balancing.

We believe the performance of the conservative protocol can be further improved by exploiting the lookahead in the partitions. Also, the speedups for both conservative and optimistic protocols will improve with bigger circuits.

Circuit partitioning was shown to have a major impact on the performance of both the parallel simulation techniques. Specifically, an acyclic partitioning algorithm was shown to have much better performance than an iterative partitioning algorithm. It was shown that the partitioning metric normally used in VLSI partitioning, namely cut-sets, is relatively ineffective as far as parallel simulation is concerned. The impact of circuit partitioning on parallel simulation is being studied in greater detail; in particular, the effect of metrics other than acyclicity will be evaluated.

References

[1] Jeffrey M. Arnold. Parallel simulation of digitial LSI circuits. Technical report no 86-295, Computer Science Department, MIT, Cambridge, MA, January 1986.

[2] R. Bagrodia, K.M. Chandy, and W-L. Liao. An experimental study on the performance of the space-time algorithm. Technical Report CSD-910061, Computer Science Dept, UCLA, Los Angeles, CA 90024, August 1991.

[3] R. Bagrodia, K.M. Chandy, and W-L. Liao. An experimental study on the performance of the space-time algorithm. In *Workshop on Parallel and Distributed Simulation*, January 1992.

[4] R. Bagrodia and Wen toh Liao. Maisie: A language for design of efficient discret-event simulations. *IEEE Transactions on Software Engineering*, April 1994.

[5] Rajvie Bagrodia, Yu an Chen, Vikas Jha, and Nicki Sonpar. Parallel Gate-level Circuit Simulatin on Shared Memory Architecutres. In *Proceedings of the IEEE Intl. Conf. on Computer-Aided Design*, October 1995.

[6] R.L. Bagrodia, Zheng Li, Vikas Jha, Yu-an Chen, and Jason Cong. Parallel logic-level simulation of vlsi circuits. In *1994 Winter Simulation Conference*, Orlando, FL, December 1994.

[7] R.L. Bagrodia and Wen-toh Liao. Transparent optimizations for optimistic simulations. In *1992 Winter Simulation Conference*, Washington D.C., December 1992.

[8] M. Bailey and L. Snyder. An empirical study of on-chip parallelism. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, 1988.

[9] J. Briner, J. Ellis, and G. Kedem. Breaking the barrier of parallel simulation of digital systems. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1991.

[10] R.E. Bryant. Simulation of Packet Communications Architecture Computer Systems. Technical Report MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[11] K.M. Chandy and R. Sherman. The conditional event approach to distributed simulation. In *Distributed Simulation Conference*, Miami, 1989.

[12] K.M. Chandy and R. Sherman. Space-time and simulation. In *Distributed Simulation Conference*, Miami, 1989.

[13] J. Cong, Zheng Li, and R. Bagrodia. Acyclic multi-way partitioning of boolean networks. In *Proceedings of ACM/IEEE Design Automation Conference*, 1994.

[14] J. Cong, Zheng Li, and R. Bagrodia. Partitioning and synchronization in parallel gate-level simulation. In *Proceedings of ACM/IEEE Design Automation Conference(submitted)*, 1995.

[15] C. Fiduccia and R. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[16] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[17] D. Jefferson. Virtual time. *ACM TOPLAS*, 7(3):404–425, July 1985.

[18] Vikas Jha and Rajive Bagrodia. Parallel implementations of Maisie using conservative algorithms. In *Winter Simulation Conference*, December 1993.

[19] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans on Computer*, C-36(1), January 1987.

[20] McBryer and P. T. Wilsey. Vast: Time warp simulation of vhdl on smp workstations. In *Proceedings of the 1994 VHDL International User's Forum*, 1994.

[21] J. Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–65, March 1986.

[22] Gregory Pfister. The yorktown simulation engine: Introduction. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, page 7.1, 1982.

[23] Sundeep Prakash and Rajive Bagrodia. Parallel Simulation of Data Parallel Programs. In *To appear in 8th Workshop on Languages and Compilers for Parallel Computing*, August 1995.

[24] Larry Soule and Anoop Gupta. An evaluation of the chandy-misra-bryant algorithm for digital logic simulation. In *6th Workshop on Parallel and Distributed Simulation (PADS92), 1992 SCS Western Simulation MultiConference on Parallel and Distributed Simulation*, pages 129–138, Newport Beach, CA, January 1992.

[25] L.P. Soule. *Parallel Logic Simulation: Centralized time and distributed-time algorithms*. PhD thesis, Computer Systems Laboratory, Stanford University, June 1992.

[26] Christian Sporrer and Herbert Bauer. Corolla partitioning for distributed logic simulation of vlsi-circuits. In *7th Workshop on Parallel and Distributed Simulation (PADS93)*, pages 85–92, San Diego, CA, May 1993.

[27] Wen-king Su and C.L. Seitz. Variants of the chandy-misra-bryant distributed simulation algorithm. In *1989 Simulation Multiconference: Distributed Simulation*, Miami, Florida, March 1989.

[28] Christopher J. Terman. Rsim - a logic-level timing simulator. In *Proceedings of the IEEE Intl. Conf. on Computer-Aided Design*, pages 437–440, October 1983.

[29] K. Wong. Statistics on logic simulation. In *Proceedings of ACM/IEEE Design Automation Conference*, 1986.