LOGIC SIMULATION ON THE MARS MULTICOMPUTER

Prathima Agrawal, Chong Hoc Hao, and Michel Remillard

Prathima Agrawal, ***Chong Hoc Hao***, *and* ***Michel Remillard*** *are associated with AT&T Bell Laboratories. Ms. Agrawal is a distinguished member of technical staff in the Computing Systems Research Laboratory at the Murray Hill, New Jersey facility, where she works on computer architecture, parallel algorithms, VLSI CAD, and fault-tolerant computing. She joined AT&T in 1978 with B.E. and M.E. degrees in electrical communication engineering from the Indian Institute of Science, Bangalore, India, and a Ph.D. in electrical engineering from the University of Southern California, Los Angeles. Mr. Hao is a supervisor in the Computer-Aided Design and Test Laboratory at the Allentown facility in Cedar Crest, Pennsylvania, where he manages test and design verification CAD development. He joined AT&T in 1979 with a Ph.D. in solid* (continued on page 35)

Design verification of large VLSI (very-large-scale integration) circuits accounts for a sizable part of design time. Simulators verify the design at various levels of abstraction. A fast and accurate simulator enables the designer to introduce a higher quality product into the marketplace early. This paper describes a logic simulator implemented using the MARS (Microprogrammable Accelerator for Rapid Simulations) multicomputer. The logic simulator, AGSIM (Accelerated Good Circuit Simulator), has been integrated into the production CAD (computer-aided design) system at AT&T Bell Laboratories. To date, more than 200 application-specific integrated circuits (ASICs) of varying complexity have been simulated with MARS. This paper presents the relevant features of the MARS architecture, and the details of the logic simulator. We present the results on 10 VLSI ASIC simulations to show its increased performance over the existing software simulator, GSIM (Good Circuit Simulator), while maintaining the same accuracy. The MARS project started with accelerated logic simulation as the primary application. However, the programmable nature of the accelerator has made several other applications possible.

Introduction

VLSI technology has advanced to permit a million transistors on a chip.[1] The trend is to advance the technology even further. In spite of advanced design methods such as silicon compilation and automatic logic synthesis, design verification of large VLSI systems is the most computation-intensive task in the entire design cycle. The vehicle used for verification is *simulation*.

Simulation is invaluable to verify a design's correctness (i.e., fault-free simulation), and to study a circuit's behavior under various

Panel 1. Terms and Acronyms in This Paper

AAU	address arithmetic unit
ACK/NORM	acknowledge/normal mode
AGSIM	Accelerated Good Circuit Simulator
ASIC	application specific integrated circuit
CAD	computer-aided design
CPU	central processor unit
DT	delay table
ED	event detector
FIFO	first-in-first-out
FL	fanout list
FOU	field operation unit
FP	pointer list
FPDL	Functional Primitive Description Language
GSIM	Good Circuit Simulator
GT	gate type
I/O	input/output
IT	input table
IVL	input vector list
LIFO	last-in-first-out
MARS	Microprogrammable Accelerator for Rapid Simulations
MBD	memory block descriptor
MCC	MARS Circuit Compiler
MD	memory detector
MOS	metal oxide semiconductor
MOTIS	MOS timing simulator
OD	oscillation detector
OEM	original equipment manufacturer
OL	output log
PE	processing element
PM	physical memory
RAM	random access memory
ROM	read-only memory
SS	signal scheduler
TT	truth table
UD	unknown address detector
VLSI	very-large-scale integration
VME	Versa Module European

delay and fault conditions (i.e., fault simulation). A simulator should be fast and accurate so a design can be completed quickly without any design errors. Undetected design errors and manufacturing defects in a VLSI chip are costly, and seriously affect the quality of the manufactured product.

A simulator is used iteratively to detect design errors that are corrected by the designer and then resimulated. Many iterations are often needed before a design is committed to layout. Software simulators implemented on general-purpose computers often are too slow to simulate large circuit designs. A single iteration may take as much as 20 to 30 hours of central processor unit (CPU) time. Supercomputers could reduce this time but tend to be expensive; using them for simulation adds to product development costs.

To circumvent the problems of inefficient simulators, special-purpose computers known as *simulation accelerators* were developed. These capture the entire simulation algorithm in hardware made for this purpose, and can speed up design simulation by as much as 100 times. Commercial accelerators have focused mainly on logic simulation.[2]

The MARS[3-4] hardware accelerator was developed to address the need for a programmable machine for logic simulation. The need for programmability arose because of past experience with commercial accelerators that needed major hardware changes even for minor algorithmic enhancements and bug fixes. The present architecture was conceived with a focus on programmability [via a programmable processor, called the processing element (PE)], and system-level reconfigurability (i.e., message-passing PEs and fully connected crossbar network). The PE also contains special circuitry to access and manipulate tables of different widths and operate on bit-fields within the same word. This feature was provided with emphasis on logic simulation where most of the data structures used are lists and tables. The concurrency provided at the PE level also enables several operations to be performed in one clock cycle.

The high degree of flexibility at the processor level and at MARS's interconnect level enables the application developer to execute a variety of algorithms. However, these are limited to applications using only integer arithmetic. Our experience suggests that algorithms needing repetitive manipulations—e.g., searches through common data structures such as lists, tables, trees, and stacks—and that operate with varied data widths, are ideally suited for MARS.

The remainder of this paper first describes MARS architecture, then elaborates on the logic simulation application, emphasizing algorithm and data partitioning. Logic simulation performance measurements taken from an operating MARS system show the degree of acceleration achieved.

Architecture

The MARS system consists of a cluster of 15 PEs interconnected with a fully connected crossbar switch. A MARS PE is a 16-bit microprogrammed processor with special features for message passing, list and table manipulation, and bit field operations. Each PE has its own local memory. Communication between PEs is performed entirely by message passing. The crossbar switch can be reconfigured each clock cycle. A host processor that can access each PE's memory communicates with the processors via interrupts.

Figure 1 shows the architecture of a MARS cluster, a ten-layer VME (Versa Module European) printed circuit board that plugs into a host workstation. Total memory in the cluster is 9 megabytes (Mbytes), unevenly distributed among the 15 PEs. This distribution was driven by the logic simulation application for supporting up to 64K logic gates. The local message switch is a 16 × 16 crossbar connecting the 15 PEs and the global communications network. The switch is 16 bits wide, and is dynamically configured. At each clock cycle, each input channel selects an output channel for the next clock cycle. An arbiter in the switch resolves conflicts and prevents messages from being interleaved.

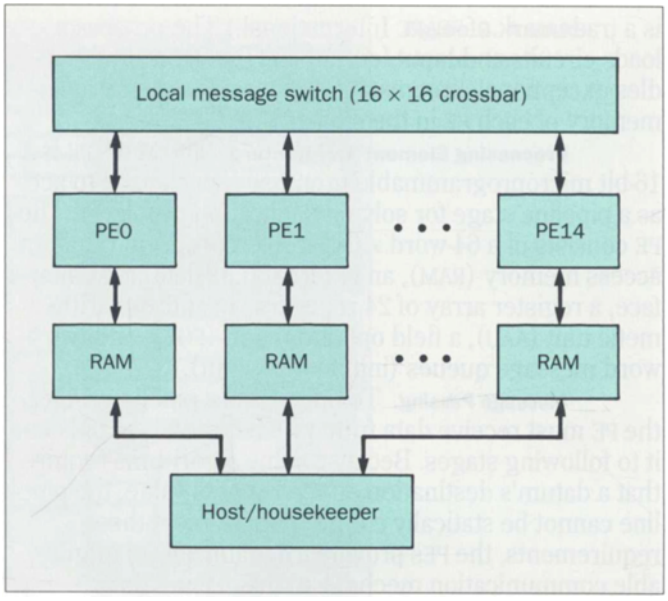

Figure 1. Cluster level architecture of MARS showing links between host/housekeeper, RAMs and PE0 through PE14, and the local message switch.

Each PE acts as one pipeline stage of the accelerated algorithm. For example, the logic simulation algorithm is partitioned into stages that do event scheduling, event cancellation, oscillation detection, signal fanout, function evaluation, delay computation, and so on. PEs communicate with each other via the local message switch. A local memory is associated with each PE. The size of this memory is 64K-words × 16 bits per word on most PEs. Some PEs have larger memories to support pipeline stages with larger storage requirements. PEs have 24-bit addresses allowing them to access up to 16 million words of local memory.

The housekeeping processor is a M68020 in the SUN-3™ Workstation and a SPARC™ processor in the SUN-4s Workstation™. (SUN Workstation [independent of model] is a trademark of Sun Microsystems, Inc.; SPARC

is a trademark of SPARC International.) The processor loads circuits and input/output (I/O) vectors, and handles exceptional circumstances. It can access the local memory of each PE in the cluster.

Processing Element Architecture. The MARS PE is a 16-bit microprogrammable processor specialized to act as a pipeline stage for solving application problems. The PE consists of a 64-word × 64-bit microprogram ramdom access memory (RAM), an external (i.e., data) RAM interface, a register array of 24 registers, an address arithmetic unit (AAU), a field operation unit (FOU), and two 4-word message queues (input and output).

Message Passing. To operate as a pipeline stage, the PE must receive data from preceding stages and send it to following stages. Because some algorithms require that a datum's destination depends on its value, the pipeline cannot be statically configured. To meet these requirements, the PEs provide a dynamically configurable communication mechanism that passes data efficiently between PEs, and synchronizes PE execution with the arrival of data.

Data transport is performed by mapping the message queues into the register file. The PE can send or receive a message simply by writing or reading a register. The destination PE is selected by writing a PE number to R14 (the destination register). Once a destination is selected, writing to R31 transmits a message word. Reading from R31 receives a message word. An interlock is provided to prevent interleaving of multiword messages sent to the same destination.

Synchronization is performed via stalls or interrupts. If a write is attempted with the output queue full, or if a read is attempted with the input queue empty, the processor will stall until it can complete the operation. Alternatively, a PE can execute a background task and arrange to be interrupted (via a trap mechanism) when the output queue is no longer full, or when the input is no longer empty.

The combination of register-mapped message passing with synchronization using stalls and interrupts provides MARS with low-overhead communication. A synchronized message may be sent in a single clock cycle. By comparison, conventional message-passing multicomputers take several cycles to send a message. Because they lack hardware support for communication and synchronization, the necessary operations use costly system calls that may take hundreds of machine cycles. The hardware support for message passing on MARS allows problems to be decomposed into much smaller units (single pipeline stages) than is possible on conventional multicomputers. This fine-grained decomposition exposes more of the parallelism inherent in the underlying problem structure.

Table and Bit Field Operations. Many of the simulation algorithms that motivated the development of MARS were dominated by table and list references, and by access to packed bit-field data structures. Table references are performed on tables of different widths. In a logic simulator, for example, the table holding the input values of each gate is eight bits wide. Each record in this table contains four 2-bit fields. Each encodes one of the four possible logic values (0, 1, X, Z). The table holding the truth table of each gate type is two bits wide. Each record contains a single logic state. The address manipulation and bit-field extraction required to access these narrow tables on a conventional processors is time-consuming.

MARS efficiently accesses tables of varying widths (in powers of 2) by performing an in-hardware address shift and bit-field extraction or insertion on every memory reference. With this access hardware, a 64K × 16 memory can be accessed in aspect ratios varying from 1M × 1 to 64X × 16. Writes to external memory in units of less than 16-bits (1, 2, 4 or 8 bits) do a read-modify-write operation, and require two clock cycles.

Many simulation algorithms require operations on packed bit fields. Packing and unpacking these fields on conventional machines is time-consuming. MARS uses a special function unit, the FOU, to do bit field operations without unpacking the data. The FOU extracts bit fields from its two source operands, operates on these bit

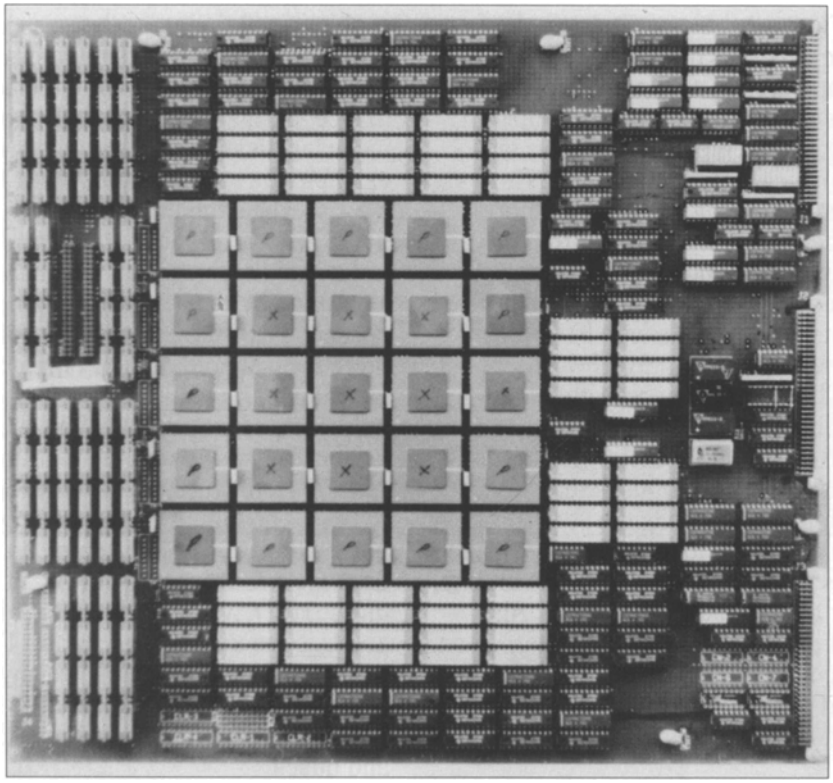

Figure 2. A MARS cluster board.

fields, and inserts the result into a bit field of its A operand. The extract, operate, and insert are all performed in one cycle.

Crossbar Switch. The PEs within a cluster are connected by a 16-input × 16-output × 16-bit crossbar switch constructed from 10 16 × 16 × 2 crossbar chipset. Eight chips—with each chip 2-bit sliced—are responsible for 16-bit data transmission. The other two provide parity and flow control functions. The same implementation is used by all 10 chips.

A source PE requests connection to a destination PE by putting the destination address on the address lines of its column. Connection is established if the destination is free; otherwise the request is blocked. If more than one PE tries to connect to the same destination, the one with the higher priority gets connected, and the others are blocked. Once made, a connection is maintained until the PE requests a new destination, or does not assert the HOLD signal and there was no data transfer in the previous cycle.

There is no storage in the cross-points. Flow control is performed with handshake lines. When the acknowledge/normal (ACK/NORM) mode line of the chip is high, the reverse lines are enabled. These lines are used to acknowledge receipt of data, or to signal that the input buffer at the destination is temporarily full. The source can monitor these signals and control the flow of data appropriately.

Cluster Board. Figure 2 shows a photograph of a MARS cluster board. The cluster is packaged on a triple-height VME board and operates as a plug-in to a SUN-3 Workstation. The cluster contains 10 crossbar chips (in the center of the board), 15 PE chips (immediately surrounding the crossbars), local memory, and a bus interface. Each PE has at least 64K 16-bit words of memory constructed from off-the-shelf 64K $\times$ 4 50ns (nanoseconds) static RAM chips. To support pipeline stages with varying memory requirements, the total memory on the board is distributed as: four PEs with 64K words, six PEs with 256K words, four PEs with 512K words, and one with 758K words. This distribution was guided by the logic simulation application of 64K logic gates.

In this system the SUN-3's 68020 (or the SUN4's SPARC) processor serves as both the housekeeper and the host. The VME bus interface permits the SUN's processor to send and receive messages to and from the PEs, to read and write each PE's memories, to set a single bit into each PE, and to monitor a single bit from each PE. An interrupt control circuit permits the SUN to be interrupted by the PEs. A statistics counter is also provided to count the clock cycles elapsed or the messages sent by any PE to aid in measuring performance.

Logic Simulation on MARS

For logic simulation, the circuit to be simulated is modeled as an interconnection of logic or boolean gates.[5] This model corresponds to a circuit graph where the nodes are logic gates with edges representing their interconnections. The nodes may have delays that represent signal propagation delays through logic gates and interconnections. A logic simulator computes the logic values at every node (i.e., the output of a logic gate) for a given set of inputs as a function of time. A multiple-delay simulator permits arbitrary delays to be assigned to the gates in the circuit. By contrast, a unit-delay simulator assumes every gate in the circuit has one unit of delay.

A family of logic simulators, ranging from a simple unit-delay to a multiple-delay logic simulator for sophisticated timing analyses, is implemented on MARS.[6] The evolution of these simulators was motivated by the need to progressively enhance accuracy and performance. In this paper, we shall show the concept of binding an application to the architecture of MARS using as an example the basic multiple-delay logic simulator.

The simulation algorithm is in well-defined partitions. The partitioning is performed according to two criteria: *load balancing* and *data access*. Load balancing is attempted by assessing how many machine cycles are needed to execute each partition of the algorithm. Because there is no shared memory in the MARS accelerator, the partitioning must also be performed so the data structures needed by each pipeline stage reside in the stage's local memory. Finally, the partitions are allocated to processing elements that form the logic simulation pipeline.

The logic simulator on the MARS hardware supports simulation of logic gates up to 64K. Each gate can have up to four inputs, and any of the four logic values: 0(00), 1(01), X(10), and U(11). X is the unknown state and U is the unconnected state. The U state represents the absence of connectivity to any of the four inputs of a primitive. It is also interpreted as the high-impedance state while simulating logic modeled for pass transistors and buses.

A block diagram of the logic simulator partitioned for implementation on MARS is shown in Figure 3. It consists of three pipelines: fanout update and evaluation, event-logging, and functional memory. Each block in the figure corresponds to a PE that is microcoded to perform the specified function. The data needed by each stage is either in the local memory or is passed from other PEs. The microprograms set the destination PE addresses and direct the crossbar switch to select the successor stage in the pipeline. PEs that need to communicate to a common PE use the channel hold facility provided in the PE, and crossbar chips to communicate without message sequencing problems.

The block-marked housekeeper represents a SUN Workstation. The housekeeper manages the

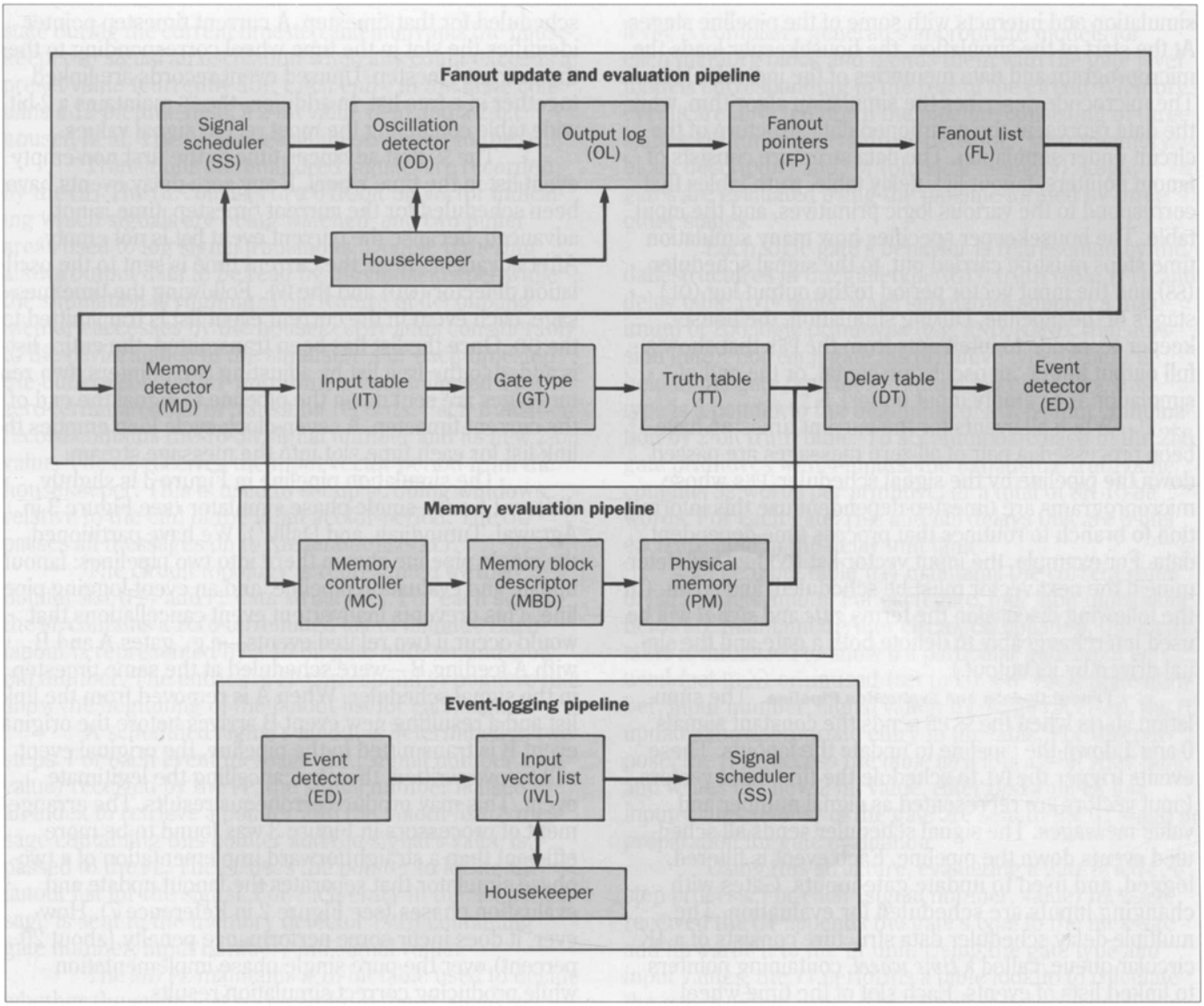

Figure 3. Schematic illustration of multiple delay logic simulation pipelines. The illustration shows event-logging, memory evaluation, fanout update pipelines and evaluation pipelines.

simulation and interacts with some of the pipeline stages. At the start of the simulation, the housekeeper loads the microprogram and data memories of the individual PEs. The microcode describes the simulation algorithm, while the data represent the partitioned data structure of the circuit under simulation. The data structure consists of fanout pointers, fanout list, delay table, truth tables that correspond to the various logic primitives, and the input table. The housekeeper specifies how many simulation time steps must be carried out, to the signal scheduler (SS) and the input vector period to the output log (OL) stages of the pipeline. During simulation, the housekeeper responds to interrupts from the PEs that show a full output buffer, an oscillating signal, or the end of simulation (i.e., empty input buffer).

When all events for the current timestep have been processed, a pair of all-zero messages are passed down the pipeline by the signal scheduler. PEs whose microprograms are timestep-dependent use this information to branch to routines that process time-dependent data. For example, the input vector list (IVL) PE will determine if the next vector must be scheduled, and so on. (In the following discussion the terms *gate* and *signal* will be used interchangeably to denote both a gate and the signal driven by its output.)

Fanout Update and Evaluation Pipeline. The simulation starts when the SS PE sends the constant signals 0 and 1 down the pipeline to update the fanouts. These events trigger the IVL to schedule the first input vectors. Input vectors are represented as signal number and value messages. The signal scheduler sends all scheduled events down the pipeline. Each event is filtered, logged, and used to update gate inputs. Gates with changing inputs are scheduled for evaluation. The multiple-delay scheduler data structure consists of a 4K circular queue, called a *time wheel*, containing pointers to linked lists of events. Each slot of the time wheel corresponds to a timestep, and identifies the event list for that timestep. Each event record consists of a 16-bit gate number, and a 16-bit pointer to the next event scheduled for that timestep. A current timestep pointer identifies the slot in the time wheel corresponding to the the current timestep. Unused event records are linked together in a free list. In addition, the SS maintains a 2-bit wide table containing the most recent signal values.

The SS first advances time to the first non-empty event list in the time wheel. If any zero-delay events have been scheduled for the current timestep, time is not advanced, because the current event list is not empty. After advancing time, the current time is sent to the oscillation detector (OD) and the IVL. Following the time message, each event in the current event list is transmitted to the OD. Once the list has been transmitted, the entire list is added to the free list by adjusting two pointers; two zero messages are sent down the pipeline to signal the end of the current timestep. A seven-clock cycle loop empties the link list for each time slot into the message stream.

The simulation pipeline in Figure 3 is slightly different from a single-phase simulator (see Figure 3 in Agrawal, Tutundjian, and Dally[7]). We have partitioned the single pipeline shown there into two pipelines: fanout update and evaluation pipeline, and an event-logging pipeline. This prevents inadvertent event cancellations that would occur if two related events—e.g., gates A and B with A feeding B—were scheduled at the same timestep in the signal scheduler. When A is removed from the link list and a resulting new event B arrives before the original event B is transmitted to the pipeline, the original event will be overwritten, thereby cancelling the legitimate event. This may produce erroneous results. The arrangement of processors in Figure 3 was found to be more efficient than a straightforward implementation of a two-phase simulator that separates the fanout update and evaluation phases (see Figure 2 in Reference 7). However, it does incur some performance penalty (about 20 percent) over the pure single phase implementation while producing correct simulation results.

The purpose of the OD stage is to detect zero-delay oscillations. The OD maintains, in a 16-bit wide table, a count of how many times each signal has changed

state during the current timestep and interrupts the housekeeper to signal an oscillation when any count exceeds a preset value (currently 15). Each entry in the table contains a 12-bit time field, a 2-bit value field and a 2-bit unused field. The OD passes all messages on to the OL.

Transitions on monitored signals are recorded by the OL. The OL consists of a 64K-bit bit vector indicating which signals are being watched, and two buffer areas used to log signal transitions. Double buffering allows output data to be read by the housekeeper while the simulation is running. This feature of decoupling memory accesses by the housekeeper adds considerably to the performance of the simulator. For each timestep the buffers contain a 1-word time stamp followed by a zero-terminated list of transition records. Each transition record contains the 16-bit signal number and its new 2-bit value. The OL receives the input vector period from the housekeeper. This is used to set up strobing windows relative to the end of the input vector period. The OL passes all messages on to the fanout list (FL).

The circuit topology is represented by the pointer list (FP) and FL data structures. For each signal the FL contains a zero-terminated list of fanouts. Each fanout is represented by a 16-bit gate number and 2-bit pin number. The entries in FP correspond to pointers that show the beginning of the fanout list for each signal.

A scheduled signal's fanout is determined in two steps. For each event message (i.e., signal number and value) received by the FP, the signal number is used as an index to retrieve a pointer into the fanout list. A message containing this pointer and the signal's value is passed to the FL. The FL uses the pointer to locate the fanout list for the signal. For each entry in the list a message is sent to the memory detector (MD) containing gate number, input number, and signal value.

The MD PE maintains a 16-bit table used to decide whether the event just received from the FL stage is a logic gate event or a functionally modeled memory gate event. RAMs and ROMs in the circuit being simulated are modeled functionally rather than at the detailed gate level. A compiler[8] generates appropriate models for each memory block and blends them with the gate level models corresponding to the rest of the circuit. Memory events are directed down the pipeline consisting of three stages: UD (unknown address detector), MBD (memory block descriptor) and PM (physical memory). Logic gates are evaluated using the pipeline formed by three other stages.

Each logic gate's behavior is represented by the data structures in the gate type (GT), truth table (TT), and delay table (DT) units. MARS simulators support a maximum of 256 logic primitives (e.g, AND, NAND, transmission gate). For each gate, its primitive type (8 bits) is held in the gate type table. Shifted left eight bits, this type is a pointer to the beginning of a 256-input combination by 2-bit truth table. To accommodate each of the 256 gate primitives with 4-inputs, the exhaustive truth table contains 32 words per primitive, or a total of 8K 16-bit words. For each gate, rise and fall delays that are 8 bits each are held in the delay unit table.

The input table (IT) maintains the current value of each gate's inputs. For each gate, four 2-bit input value fields are maintained. After circuit compilation, the input table is initialized to show if a particular gate input is used (set to X) or unused (set to U). For each (gate number, input number, signal value) message received, the IT updates the appropriate entry in the table. For this purpose, the IT PE access the table as if it is a 2-bit wide table and writes the new 2-bit value. After updating all four input values (8-bits) of the gate are sent to the GT stage in preparation for gate evaluation.

Using this structure, evaluating a gate is a three step process. For each (signal number, value) message received the GT appends the gate's type to the message and forwards it to the TT unit. Using the gate type and input values, the TT performs a table lookup to calculate the gate's output value. The TT unit passes a (signal number, value) message. The DT receives this message and looks up the appropriate delay for the signal. Generally, the DT entries consist of rise and fall delay values (eight

bits each). The DT looks up the proper rise or fall delay depending on the new value of the signal. The MARS multiple-delay simulator uses rise and fall delays obtained by device characterization and a capacitive analysis of the circuit simulated.[9] Each of the rise and fall delays is an 8-bit integer that permits one 16-bit word to represent both delays for each gate. A (signal number, value, delay) message is forwarded to the event detector (ED) stage. The DT for a unit-delay simulator can be simplified as the delays will be either 0 or 1.

The ED works in two modes: event detection and event transmission. During the fanout update and evaluation phase, the ED filters evaluations, logging only those events that change the value of a signal. The data structure of the ED PE is a 16 bit-wide table. The event-logging auxiliary pipeline stages are described next. On receiving a pair of consecutive all-zero messages from the signal scheduler, the ED switches to the event-logging mode.

Event-Logging Pipeline. The event_logging pipeline contains three PEs: ED, IVL, and SS. The ED PE sends the events that were collected during fanout update and evaluation phase to the IVL. When all events are transmitted it sends out an all-zero message to the IVL.

The IVL inserts into the message stream additional events representing primary input signal changes at the appropriate time. Its data structure is similar to an OL buffer. It consists of a series of time frames. Each time frame contains a time stamp indicating the next time when an input vector needs to be scheduled, followed by a zero-terminated list of transition records. Each record specifies the primary 16-bit input gate number followed by a 2-bit value field. The IVL simply passes the (signal number, value, [current time + delay]) messages to the SS until a zero message is received. The IVL then inserts any input events for the current timestep into the message stream.

During the event-logging phase, the SS receives (signal number, value, delay) messages corresponding to evaluated gates that generated new events. The event list for the appropriate time slot is identified by adding delay to the current timestep pointer *mod* 4096. The signal number is then entered into an event record removed from the head of the free list. This record is then linked onto the appropriate event list. When two consecutive zero messages are received, the SS switches to the fanout update and evaluation phase.

The SS uses only seven clock cycles for processing a received event. These cycles include receiving the event message [2 message words: gate number (16-bits), value (2-bits) and event time (12-bits)], inserting it into the link list at the appropriate time slot, and adjusting the free list pointers. There is also implicit event cancellation of spurious events during this loop because the SS processes the events in the link list in last-in-first-out (LIFO) order. The link list contains only the gate numbers of scheduled events; values are stored separately in a 2-bit width table. By overwriting the value field only the latest events become valid.

Memory Evaluation Pipeline. Details of the functional memory pipeline composed of three PEs is discussed in Agrawal, Moturu, and Tutundjian,[10] and will not be repeated here. These stages make extensive use of the PEs' variable width table and bit field manipulation features.

Logic Simulation Performance. The MARS logic simulator is now in use in AT&T Bell Laboratories, AT&T's Microelectronic Centers, and some AT&T original equipment manufacturer (OEM) customer premises. More than 200 VLSI chips of varying complexity have been successfully simulated. For performance comparisons, a set of 10 typical standard cell designs are chosen. These circuits are simulated using the production version of the software simulator, MOTIS[11] [metal oxide semiconductor (MOS) Timing Simulator] as well as the multiple-delay simulator implemented in MARS. GSIM, a version of the MOTIS simulator, was used in this comparative study. The production version of the MARS simulator is known as AGSIM (Accelerated GSIM).

The data structure corresponding to each circuit to be simulated is generated with a preprocessor, MCC (MARS Circuit Compiler),[8] that uses the same input and

Table I. Characteristics of VLSI Circuits Simulated

Circuit	Total No. of Transistors	No. of Logic Transistors	No. of Gates	No. of Vectors	Size of Memory (Bits)
CKT1	198,306	53,154	23,921	4,821	24,192
CKT2	171,394	45,442	17,862	1,107	29,992
CKT3	456,579	14,211	14,348	24,816	73,728
CKT4	196,454	151,718	56,076	22,242	22,242
CKT5	37,190	37,190	18,826	65,660	0
CKT6	225,156	3,972	1,597	3,588	36,864
CKT7	215,932	19,324	9,253	2,551	32,768
CKT8	7,905	7,905	3,446	1,117	0
CKT9	154,480	101,296	45,654	3,990	8,864
CKT10	34,521	34,521	17,101	19,432	0

connectivity descriptions used by the GSIM simulator. This makes AGSIM transparent to the designer accustomed to working with GSIM. MCC extracts the logic descriptions suitable for MARS from the transistor-level data structures generated for GSIM. The logic descriptions are the various tables required by the different stages of the logic simulation pipeline described earlier.

The test circuits simulated ranged in complexity from less than 8K transistors to more than 450K transistors. But transistor count is a poor indicator of the simulation needs of the circuit. That is because large memory circuits can be simulated effectively with high-level memory primitives that need little overhead for both compilation and simulation. A more telling indicator of simulation needs is the number of random logic gates. (*Gates* are defined for this purpose as the number of 4 input logic gates needed to represent the circuit, in addition to high level memory description.) The number of gates for the test circuits ranged from 1.5K to 56K. Table I gives the characteristics of 10 production VLSI circuits from Allentown-Cedar Crest simulated on AGSIM. It lists, for each circuit, total transistor count (many transistors contribute to random logic); size of on-chip memory (in bits); and the number of vectors used in the simulations.

How many vectors are simulated is not as important as the amount of activity the vectors generated. We define circuit activity as the ratio of the total events vectors. Circuit activity was monitored by counting how many events were produced during simulation. A serial logic simulator's performance is determined by the circuit activity and I/O time consumed in activities such as processing the monitored signals and vector injection. These two quantities will thus determine the simulation complexity. Our experience showed that the amount of speedup obtained with the MARS accelerator depends only on the circuit activity. The latter is true because the

Table II. Measured Simulation Complexity and Performance

Circuit	Total Events (Millions)	Ave. Activity (Events/vector)	Time in Seconds AGSIM(elapsed)	GSIM(CPU)	Speedup
CKT1	62.8	13026	460	71,131	155
CKT2	4.16	3758	65	3433	53
CKT3	176.5	7112	1,141	134,195	118
CKT4	551.8	24808	4,111	497,881	121
CKT5	144.3	2197	901	144,411	160
CKT6	1.18	329	115	1,100	9
CKT7	1.66	650	212	12,041	57
CKT8	0.34	304	68	353	5
CKT9	39.6	9924	281	32,162	114
CKT10	192.1	9886	1,134	191,503	169

OL stage of the simulation pipeline in Figure 3 is designed so the main simulation activities of event update and evaluation are undisturbed because of monitoring and printing the signals selected. Double buffering the OL stage's memory allows the housekeeper processor to empty one buffer while computation proceeds by a buffer switch. This incurs the least I/O penalty. Without this feature, the simulator must frequently fill and empty the buffers if many gates have to be monitored continuously. A larger circuit activity tends to keep the simulation pipeline full, thereby increasing the performance.

Table II shows—for each of our 10 benchmark circuits—the measured simulation times for both GSIM and AGSIM, and the circuit activity in events per vector. The times measured from AGSIM are the *elapsed* times; this is the only time measurement facility available on the system. Also shown in the table are the the CPU times on a SUN-3/260 for GSIM. GSIM simulation of the same circuits on a SUN-4/260 were faster on the average by a factor of two over SUN-3/260. However, there was no significant improvement for AGSIM when the MARS board used a SUN-4/260 as plug-in platform. The SUN Workstations were configured as standalone machines with greater than minimum memory configurations. Our measurements on the hardware show an average performance improvement of about 100 times over MOTIS GSIM. Maximum speedup is achieved by circuits with a large activity and many events simulated. The increased speedup for higher activity circuits producing many events results from keeping the pipeline busy compared to the cycles lost because of pipeline latency and other I/O-related overheads. For less active circuits, the overheads mentioned above become the dominant factor pulling the performance down. The small performance increase in these cases merely represents the ratio of the

overheads for GSIM and AGSIM. AGSIM with a carefully designed hardware pipeline incurs a smaller overhead. Circuits CKT1, CKT3, CKT4 CKT5 CKT9 and CKT10 generate many events with a higher activity per vector period. All other circuits generate fewer events, typically less than 5 million. Circuits CKT2 and CKT10, in spite of simulating almost as many logic gates, produce vastly differing speedups. This is because of the larger number of events produced by CKT10 (192 million) compared to 4 million produced by CKT2. The worst performance is obtained for CKT8 with the least number of events processed. These circuits, CKT6 and CKT7, have low activity per vector. CKT2 with a larger activity per vector also exhibits poor performance. This may be because of an uneven distribution of activity during simulation resulting in bursts of high activity periods separated by periods of low or no activity.

It is important to remember we are comparing elapsed time of MARS with the CPU time on SUN-3/260 for GSIM. Hence, the speedup figures quoted are underestimated. We cannot measure the gain in performance of the simulator on a pipelined architecture with many PEs compared to the entire simulator on a single PE. Individual PEs program and data memory spaces are not large enough to accommodate the entire simulator.

The three pipelines were carefully designed to minimize the idle time of each PE resulting in high processor utilization. The time taken for processing all events in a time step is large compared to the pipeline latency: for logic simulation, this is 60 cycles. In our simulation pipeline an event takes an average of eight cycles to process. In this computation, we assume an average fanout of two per gate. Also, the event-logging and vector injection take about four clock cycles. The critical part of the fanout update and evaluation pipeline is from the FL to the ED stages (see Figure 3). Every one of the stages in this pipeline is designed to complete processing of an incoming event in four cycles. All other stages not in this critical subpipeline require four to seven cycles. Thus our decomposition of the logic simulation algorithm results in an even load among the processors. The subpipeline parallel to the logic evaluation is used for memory evaluation. Although a memory related event is costlier to process than a logic-related event, the former is less frequent.[6]

A system of simulators that range from the multiple-delay logic simulator to a mixed-mode simulator that includes standard cell logic, behavior (user specified C-blocks), functional memory described in FPDL (Functional Primitive Description Language) and standard cell C-models, is being developed. Including behavioral modeling in MARS without hardware change or additional hardware demonstrates the power of programmability of the accelerator. This contrasts with other simulation accelerators that need additional hardware.[12]

Fault Simulation

Fault simulation[5,13] has been a challenge to MARS mainly because of the limited memory (9 Mbytes) available on a MARS cluster board. This forced us to experiment with a spectrum of fault simulators ranging from the simplest, slowest, and least memory-intensive serial fault simulator, to the fastest, most complicated, and memory-intensive concurrent fault simulator.[14] The concurrent fault simulation algorithm is the most efficient of all currently known fault simulation techniques. Here $n + 1$ simulations proceed concurrently, where n is the total faults being simulated. In addition to the n fault simulations, the fault-free simulation proceeds concurrently. The simulator maintains the status of the fault-free as well as faulty circuits. However, details corresponding to a faulty circuit are maintained only if the faulty circuit's behavior differs from the fault-free circuit's. This enables the simulator to process many faults at the same time. Even then, the memory needed for maintaining this information is prohibitive for large circuits.

Due to the limited memory in the MARS system, we devised a multiple pass concurrent fault simulation algorithm. The number of faults processed per pass is precomputed, and fixed size fault lists are set up so these

faults will be simulated regardless of how many fault effects they produce. Thus, the need for dynamic memory management found in conventional concurrent fault simulators is avoided. Multiple passes will process the entire fault list. Experiments on large circuits show that a single fault simulation pass using MARS is about two orders of magnitude faster than the software MOTIS fault simulator running on a SUN-3/260, for the same number of faults. Fault simulation uses the same set of data structures as a logic simulator to represent the circuit topology, and to process good and faulty circuit-related events. Fault lists are maintained as singly linked lists at every gate in the circuit. Depending on circuit activity, this list must be updated at each timestep: this is the predominant performance bottleneck in any fault simulator. By implementing the concurrent fault simulation algorithm on MARS, we have shown the strength of our architecture for supporting a complex application. Details of algorithms, data structures, and performance results for this and a serial fault simulator implemented on MARS, can be found in Agrawal, Agrawal, et al.[15]

Conclusion

We have built a programmable accelerator, MARS, that achieves performance comparable to special purpose simulation engines, while providing the flexibility of a programmable machine. Problems well-suited to this architecture have a high degree of functional parallelism. In particular, these include applications requiring extensive manipulation of tables of different widths, singly and doubly-linked lists, and LIFO and FIFO (first-in-first-out) queues. This is the main reason for the efficient implementation of logic simulation on MARS. Furthermore, the concurrent fault simulation algorithm involves an efficient fault list search algorithm. Other applications programmed on MARS recognize preprocessed speech using a beam search technique,[16] switch-level simulation,[7] and a maximum flow computation problem. The main reason for successfully implementing such vastly different applications on MARS is in recognizing that the underlying algorithms in each of these is related to manipulation of graph-based data structures. Indeed, logic simulation also belongs to this class of graph-based algorithms.

In the restricted single-cluster implementation, MARS may not be most efficient for multiprocessing an algorithm where simultaneous processing of different data sets is required. This is because the 64-word program RAM severely limits the complexity of each PE's microprogram. This limitation also affects building complex pipelined algorithms when each pipeline stage needs to use more than 64 microwords of program memory. The present MARS system is clocked at 5MHz (megahertz). Higher clock rate will proportionately improve the performance.

An area for further research is developing programming tools for heavily pipelined architectures like MARS. The applications cited in this paper were programmed using a microassembler. Separate microprograms were written for each pipeline stage. Recently a PE-level optimizing compiler, that permits translation from C to a microprogram, has been implemented.[17] A high-level programming environment with efficient and automatic pipelining is being implemented.

Acknowledgments

Several people have contributed to the successful development of the MARS accelerator. Thanks to W. J. Dally of MIT for his valuable contributions to several aspects of this project. We acknowledge W. C. Fischer, A. S. Krishnakumar, H. V. Jagadish, R. V. Gunther, and T. Misawa for their efforts in building the hardware. Thanks to V. D. Agrawal, S. Chatterjee, K-T. Cheng, C. S. Moturu L. A. Noronha, and R. Tutundjian in developing algorithms and their implementations on the MARS hardware. Members of the Test and Design Verification Group have been instrumental in developing MARS into a production quality tool. In addition, several colleagues at AT&T Bell Laboratories helped with processing, testing, and packaging of the VLSI chips in the MARS system. Special thanks to Ismail Eldumiati, Chong Lee, Lincoln

Pierce, Yash Punati, and Clay Schneider, who risked using the first version of the AGSIM simulator for verifying their designs. Without their cooperation and valuable feedback, AGSIM would not be in its present state.

References

1. T. S. Perry, "Intel's Secret Is Out," *IEEE Spectrum*, Vol 26, No. 4, April 1989, pp. 22-28.
2. T. Blank, "A Survey of Hardware Accelerators Used in Computer-Aided Design," *IEEE Design and Test of Computers*, Vol. 1, August 1984, pp. 21-39.
3. P. Agrawal, W. J. Dally, A. K. Ezzat, W. C. Fischer, H. V. Jagadish, and A. S. Krishnakumar, "Architecture and Design of the MARS Hardware Accelerator," *Proceedings of the 24th Design Automation Conference*, Miami Beach, Florida, June 29-July 1, 1987, pp. 101-107.
4. P. Agrawal, W. J. Dally, W. C. Fischer, H. V. Jagadish, A. S. Krishnakumar, and R. Tutundjian, "MARS: A Multiprocessor-Based Programmable Accelerator," *IEEE Design and Test of Computers*, Vol. 4, October 1987, pp. 28-36.
5. H. Fujiwara, *Logic Testing and Design for Testability*, MIT Press, Cambridge, Massachusetts, 1985.
6. P. Agrawal and W. J. Dally, "A Hardware Logic Simulation System," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, Vol. 9, No. 1, January 1990, pp. 19-29.
7. P. Agrawal, R. Tutundjian, and W. J. Dally, "Algorithms for Accuracy Enhancement in a Hardware Logic Simulator," *Proceedings of the ACM/IEEE Design Automation Conference*, Las Vegas, Nevada, June 25-29, 1989, pp. 645-648.
8. P. Agrawal and L. W. Noronha, "Modeling Circuits in the MARS Hardware Accelerator," *Proceedings of the IEEE International Conference on Computer-Aided Design*, Santa Clara, California, November 9-12, 1987, pp. 492-495.
9. F. C. Chang, C. F. Chen, and P. Subrahmaniam, "An Accurate and Efficient Gate Level Delay Calculation for MOS Circuits," *Proceedings of the 25th ACM/IEEE Design Automation Conference*, Anaheim, California, June 1988, pp. 282-287.
10. P. Agrawal, C. S. Moturu, and R. Tutundian, "Modeling and Simulation of Functional Memory Blocks in the MARS Accelerator," *Proceedings of the International Workshop on Hardware Accelerators*, Oxford, U.K., September 20-21, 1989.
11. C. F. Chen, C. Y. Lo, H. N. Nham, and P. Subrahmaniam, "The Second Generation MOTIS Mixed-Mode Simulator," *Proceedings of the 21st ACM/IEEE Design Automation Conference*, Albuquerque, New Mexico, June 1984, pp. 10-17.
12. R. Milne, "Hardware Simulator Takes On Behavioral Simulation," *Electronics Design*, Vol. 37, No. 11, May 25, 1989, pp. 81-82.
13. D. K. Pradhan, *Fault-Tolerant Computing: Theory and Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
14. E. G. Ulrich and T. Baker, "Concurrent Simulation of Nearly Identical Digital Networks," *Computer*, Vol. 7, No. 4, April 1974, pp. 39-44.
15. P. Agrawal, V. D. Agrawal, K. T. Cheng, and R. Tutundjian, "Fault Simulation in a Pipelined Multiprocessor System," *Proceedings of the IEEE International Test Conference*, Washington, D.C., August 29-31, 1989, pp. 727-734.
16. S. Chatterjee and P. Agrawal, "Connected Speech Recognition on a Multiple Processor Pipeline," *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, Glasgow, Scotland, May 23-26, 1989, pp. 774-777.
17. P. H. Kenyon, P. Agrawal, and S. C. Seth, "High-Level Microprogramming: An Optimizing C Compiler for a Processing Element of a CAD Accelerator," *Proceedings of the 23rd International Symposium and Workshop on Microprogramming and Microarchitecture*, Orlando, Florida, November 27-29, 1990.

Biographies (continued)

state physics from Indiana University, Bloomington. Mr. Remillard is a member of technical staff in the VLSI Design Laboratory, also in Cedar Crest, where he coordinates and plans the AT&T Design System (ADS) CAD package for VLSI design. He joined AT&T in 1982 with a B.S.E.E. in electronic devices from the University of Texas at Austin, and an M.S.E.E. in VLSI design for digital services from the University of California, Berkeley.

(Manuscript received November 1, 1990)