Fast Test Simulation via Distributed Computing

Kai-Hui Chang, Jeh-Yen Kang, Chi-Lai Huang, John P. Hayes and Igor L. Markov

AVERY-TR-001-06

July 1, 2006

Avery Design Systems
Andover, MA 01810
USA

Fast Test Simulation via Distributed Computing

Kai-Hui Chang†, Jeh-Yen Kang‡, Chi-Lai Huang‡, John P. Hayes† and Igor L. Markov†
†University of Michigan, Ann Arbor, MI 48109
‡Avery Design Systems, Andover, MA 01810
†{changkh, jhayes, imarkov}@umich.edu, ‡{kankan, clhuang}@avery-design.com

Andover, MA 01810
USA July 1, 2006

Abstract

Parallel processing techniques have been applied to logic simulation in the past with limited success. An aspect of this problem that has received little research attention is the impact on performance of application-specific simulation of test sequences. We approach the problem from this angle and identify an important practical application, simulation of manufacturing test patterns for ICs, which can be successfully parallelized. This is due to the high and balanced circuit activity that is typical of manufacturing tests. To exploit these characteristics, we develop several new techniques to reduce communication overhead and balance workload. We also describe a circuit partitioning methodology based on these techniques, and evaluate the impact of hypergraph partitioning algorithms. Our empirical results show that the proposed approach can significantly accelerate parallel simulation of manufacturing tests.

1 Introduction

Testing is an essential and often costly step needed to ensure that integrated circuits (ICs) are manufactured free of defects. In order to reduce testing time, test patterns that can detect more defects in less time are preferred, and automatic test pattern generation (ATPG) techniques are often used to generate these patterns. After the test patterns are generated, they usually need to be processed by a logic simulator to calculate the output responses and fault coverage. It is important to note that long sequential input patterns that must be applied through many cycles typically cannot be easily split into smaller portions. As circuit size increases, it takes longer to simulate these test patterns – sometimes days or even weeks – which is painfully long and increases the product's time to market.

Distributed parallel simulation is one way to solve this problem, and it has been studied for more than 20 years [1, 6, 10, 13, 15, 18]. Many important issues in parallel simulation such as system architecture, data partitioning, and workload balancing were studied extensively in the early 90s, but relatively few new ideas have emerged since then. For example, it is well known that for the best performance using parallel simulation, the workload should be partitioned among the processors in a balanced way, and communication between processors should be minimal. However, parallel logic simulation has never become popular despite all the past research efforts and the increasing availability of large networks of workstations that can implement distributed parallel processing. We summarize the main obstacles to adopting parallel simulation as follows:

- It is, in general, difficult to estimate workload without actually running simulation, which makes the partitioning a challenging task. Although structural partitioning is a well-studied problem, there is little research on the effect of dynamic, application-specific workloads in the form of *test vectors* on the performance of competitive parallel simulators running on the state-of-the-art hardware.
- Most previous parallel simulation systems are implemented by modifying publicly available simulators or by creating their own simulation engines, which cannot take advantage of the improvements from commercial simulators. For example, the Icarus simulator used in [12] is more than 10X slower than commercial simulators as reported in [25]. As a result, serial commercial simulators can often easily outperform those parallel simulation systems and make them impractical.

In order the address the first problem, we study the impact of test pattern selection on the performance of parallel simulation. As we demonstrate, the test pattern sets used after IC manufacture, which are ATPG-based tests targeted at the industry-standard stuck-at-0/1 fault model, have some characteristics that make them especially suitable for parallel simulation. In particular, they form a workload that is often much

more balanced than that of functional test sets. Furthermore, interprocessor communication can be reduced or localized with clever partitioning and synchronization methods. In this paper, we investigate ways to exploit these features of manufacturing test sets, including the integration of a balanced min-cut partitioner into our partitioning flow, and build our parallel system upon commercial simulators [20]. We demonstrate successful speed-up of simulation for several large IC designs.

The rest of the paper is organized as follows. Relevant background concerning ATPG and parallel simulation is given in Section 2. Section 3 identifies important characteristics of manufacturing tests, and Section 4 proposes new techniques for parallel simulation of these tests. This technique is empirically validated in Section 5, where we also evaluate the significance of hypergraph partitioning and network latency to parallel simulation. Section 6 concludes the paper.

2 Background

We first briefly review relevant concepts in test generation and then discuss distributed parallel simulation.

2.1 ATPG and Fault Simulation

The purpose of functional testing is to verify that a circuit carries out its intended functions; for instance, a multiplier should perform multiplication. On the other hand, manufacturing testing verifies that the circuit does not have manufacturing defects by focusing on circuit structure rather than functional behavior. A typical example of a manufacturing defect is an open or short circuit due to dust particles. Such a defect usually has a logic or timing effect on the circuit behavior, and there are several well-known fault models to represent these defects. The stuck-at fault model is currently the most common. In this model, stuck-at-0 represents a signal that is permanently low, while stuck-at-1 represents a signal that is always high. With a suitable fault model, the test coverage of a set of test patterns can be measured. A good set of test patterns should provide high test coverage with a small number of patterns.

In order to produce high-quality test patterns, several algorithms have been proposed for automatic pattern generation, such as PODEM and FAN [4]. These techniques are effective for large combinational circuits, and they can be extended to large sequential circuits using scan chain. The basic idea behind scan design is to employ sequential elements that have a serial shift capability so that they can be connected to form long shift registers called scan chains. The scan chain elements, as shown in Figure 1, can then operate like primary inputs or outputs during testing and greatly enhance the controllability and observability of internal signals. Two modes are used for scan testing: scan mode and normal mode. In scan mode, a new test pattern and the output response of the previous pattern are shifted into and out of the scan chain, respectively. In normal mode, the pattern in the scan chain is fed into the combinational logic, and output responses are collected. These two modes are alternated to accomplish testing. In practice, these patterns may be split into several sequences so that each sequence can fit into automated test equipment's (ATE) memory and be finished within time limitation. Although simulation of each sequence can be carried out simultaneously, patterns in the same sequence still need to be simulated sequentially.

After patterns are generated, fault simulation is used to measure fault coverage and calculate output response, etc. After timing analysis is finished, simulation may need to be carried out several times using different environment configurations to make sure the patterns are valid under all circumstances. For example, four-corner simulation may need to be run at worst-case voltage and temperature extremes. These simulations are typically time-consuming due to the complicated timing model used, especially when the tests are long. It is common in industry that simulating one of the long patterns can take several days or

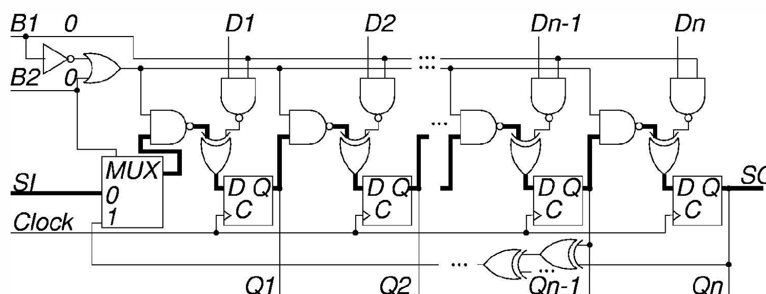

Figure 1: The BILBO general-purpose scan-chain element.

even weeks. In addition to these situations, fault simulation also needs to be repeated after each netlist modification in order to reevaluate fault coverage and recalculate new output responses.

ATPG and fault simulation generally become slower as circuit size increases. As a result, parallel ATPG has been proposed to reduce ATPG run time. The techniques used in parallel ATPG fall into five major categories [9]: fault partitioning, heuristic parallelization, search-space partitioning, functional (algorithmic) partitioning, and topological partitioning. Fault partitioning distributes different faults to different processors. Heuristic parallelization uses different heuristics on different processors for the same fault. Search-space partitioning divides the search space into disjoint pieces and evaluates them concurrently. Functional partitioning divides an ATPG algorithm into independent subtasks that can be executed on separate processors in parallel, such as running pattern generators and fault simulators on different processors [2]. Topological partitioning splits a circuit into smaller pieces. Due to the large communication overhead between these pieces, it is ineffective for pattern generation and has not become popular. However, as we will show in the next section, this method is suitable for fault simulation.

Our work focuses on simulation of manufacturing test patterns for large IC designs that employ scan chains. By manufacturing tests, we mean ATPG-based tests targeted at the industry-standard stuck-at-0/1 fault model.

2.2 Distributed Parallel Simulation

To parallelize logic simulation, the given design, which in our case is written in Verilog, is divided into pieces called partitions. These partitions are then distributed to the available processors, typically a multiprocessor computer or a network of workstations. SimCluster [20] is the distributed simulator used in our work. It is compatible with most commercial simulators and can take advantage of the efficiency of these simulators as well as any future improvements in simulation such as [14]. It employs a partition called the ”Master”, which handles signal transfers and synchronization among the partitions. All other partitions are called ”children” and are coordinated by the Master. The Master is also a Verilog simulator and can simulate part of the design. The pseudo code of SimCluster's algorithm for the Master is given in Figure 2, and the algorithm for the child is given in Figure 3.

Parallel processing is performed in lines 1-4 of Figure 3, where all the children simulate their current circuit partitions concurrently so that workload is distributed to multiple processors. There is synchronization overhead in line 5-7 of Figure 2 and 3, where the synchronization signals are sent, and the next event time is collected and sent back to each child. Additional overhead comes from propagating port changes from one child to another. This is because in distributed simulation it takes much longer to propagate port changes to other partitions. For example, if TCP/IP is used as the communication architecture, socket reads/writes

```
1 do
2   Get children's port changes;
3   Send port changes to children;
4 until no more ports change;
5 Ask children to send next event times;
6 Get childrens' next event times;
7 Send the minimum time to all children;
8 Goto 1;
```

Figure 2: SimCluster's algorithm for the Master.

```
1 do
2   Update port values from Master;
3   Send port changes to Master;
4 until Master asks for next event time;
5 Send next event time;
6 Get next event time t from Master;
7 Advance to time t;
8 Goto 1;
```

Figure 3: SimCluster's algorithm for the child.

need to be used to transfer data between partitions, and the latency is not negligible.

For better parallel processing performance, the workload shared among processors should be as balanced as possible to maximize parallelism, and communication overhead should be as small as possible to reduce latency. Partitioning is the key to achieve these goals. Therefore, some important partitioning techniques are discussed below [1].

The simplest way to partition a design is to assign gates randomly and equally to all partitions. If the workloads of any two gates are similar, this method ensures good workload balance. For better workload balance, Manjikian and Loucks [13] have the processors do presimulation to get a better estimate of the workload. Nicol and Reynold [16] also propose a method to dynamically balance the workload.

Communication and synchronization overhead is another major factor that affects the performance of distributed simulation, and several algorithms have been proposed to reduce these overheads. Workload balance is often taken care of in these algorithms by assigning similar numbers of gates to all partitions. Several important techniques to reduce communication overhead are discussed next.

Levendel et al. [11] present a partitioning method based on input/output strings. Their algorithm traces a primary input to fanout gates and selects one of them to add it to the current string. This process continues until a primary output is reached. The gates on the string are then placed in the same partition. If any gates remain unassigned, a gate is selected randomly to start a new string until all the gates are partitioned. This algorithm ensures that at least one fanout gate will be on the current processor, but it does not reduce the communication between closely related components significantly.

Smith et al. [17] employ fanin and fanout cones to reduce communication. The cone of a gate consists of the set of all gates affected by its output. The algorithm first builds cones for every gate, and the gates driven by primary inputs are assigned evenly to partitions. After the primary inputs have been assigned, a gate is randomly selected. This gate is added to the partition that has the largest number of cones containing the gate, unless that partition is already full. Mueller-Thuns et al. [15] extend the method of [17] by putting an entire cone in the same partition to reduce communication overhead. Gates may need to be duplicated in

Benchmark	Number of wires	Wire values changed per cycle		
		Random	Scan(s)	Scan(p)
B14	51924	1086	9010	13793
B17	250804	5311	6177	46567
B22	163679	2883	5233	47995
Average	100%	1.99%	7.67%	24.82%

Table 1: Comparison of circuit activity using random and ATPG-based test patterns.

different partitions, which causes some computation redundancy.

Chang et al. [6] propose a partitioner that partitions gates along flip-flop boundaries so that synchronization is reduced to the clock edges only while the timing accuracy is preserved. The partitioner runs fast and is attractive for variable-delay simulations. However, it only works if such boundaries exist, which limits its applicability.

In addition to partitioning, Kormicki et al. [10] discuss several important factors that influence the performance of distributed simulation using a network of workstations. They observed that higher activity and larger circuit size increase the speed-ups gained from parallel simulation, an observation also supported by our experimental results. However, their observation that random partitioning works almost as well as heuristic partitioning, which is also reported by Smith et al. [18], is different from ours. As we will show in our experimental results, by integrating balanced min-cut techniques into our partitioner, the speed-up is boosted by 2.45X compared with our naive heuristic partitioning, which according to Briner [3] is better than random partitioning.

3 Circuit Activity During Testing

Manufacturing testing has several features that differ from those of functional testing. These features are analyzed empirically and analytically in this section.

3.1 Empirical Observations

In order to analyze the characteristics of manufacturing tests empirically, we measured the circuit activity of three ITC99 benchmarks, including B14, B17 and B22. In this experiment, we estimate the average circuit activity in a given circuit by sampling switching activity over 20% of all wires during 100-1000 simulation cycles. Since functional tests are not available, we use random inputs instead, which should have a higher activity than functional tests. Serial mode and parallel mode scan tests are both used in this experiment, where serial mode is denoted with (s) and parallel mode is denoted with (p). From the experimental results, we can see that manufacturing tests produce much more activity than random tests, especially in parallel mode, which results in a high workload-to-communication ratio. This suggests that manufacturing tests are indeed suitable for parallel simulation. The results of this experiment are summarized in Table 1.

3.2 Analytical Observations

We also analyzed manufacturing tests analytically, and their characteristics are summarized as follows:

1. **The activity of the circuit is higher.** To detect stuck-at faults on a wire, its value must be set to both 0 and 1 at least once. Good manufacturing test sets tend to detect as many faults as possible using

few test patterns, which implies that the signal values of many wires change frequently. On the other hand, functional tests often focus on exercising a portion of the circuit and most of the circuit may be idle. Furthermore, there are often some restrictions on functional test patterns which lower circuit activity levels. For example, an invalid op-code cannot be fed to a CPU during functional testing, but there are no such restrictions on manufacturing test – as long as the op-code can detect defects, it is a good test pattern.

Wang and Gupta [19] observe that the lack or correlation between consecutive tests vectors increases circuit activity, whereas consecutive vectors applied in "normal" operation have high correlation and so cause low activity. Since functional tests inputs are basically normal inputs that are chosen to exercise certain behaviors of the circuit, they can also be expected to produce the same low circuit activity. On the other hand, there is usually very low correlation between manufacturing test patterns, thus circuit activity should be higher.

This analysis is also supported by our empirical observations. We have observed that 1.99% of the signals have their values changed in a cycle using random inputs, while 7.67% are changed in serial mode scan tests and 24.82% are changed in parallel mode scan tests. For comparison, often fewer than 3% are changed in functional tests, as reported in [1].

2. **The distribution of circuit activity is more uniform.** Manufacturing test patterns try to exercise as much logic as possible simultaneously, which distributes the circuit activity more uniformly. In scan-based designs, the scan chains also act like random input sources to the combinational logic inside the chip during scan-in and scan-out. The reason is that the outputs of the scan registers are inputs to the combinational logic. Since the pattern being shifted is usually composed of alternating 0s and 1s, it behaves like random inputs to the combinational logic, which tend to create an equally distributed workload within the chip.

As mentioned in the previous section, designs suitable for parallel simulation should have high activity between synchronization points and a balanced workload distribution. Our analysis and empirical results suggest that manufacturing test simulation is a good candidate for parallel simulation. In the next section we propose several techniques that exploit these characteristics to speed-up parallel simulation of manufacturing test patterns.

4 Techniques for Accelerating Simulation

In this section, we propose several techniques to reduce communication overhead and balance workload based on the observations given in the previous section. We develop a new partitioning methodology that targets parallel simulation of manufacturing test patterns in circuits with scan chains.

4.1 Reducing Communication Overhead

The choice of circuit timing model has a significant impact on the performance of parallel simulation, because processes must synchronize every time an event occurs, and synchronization contributes to a large portion of the communication overhead. There are three commonly used timing models. The *zero delay model* assumes that all gate and wire delays are 0. In the *unit delay model*, it is assumed that all gate delays are 1. The *variable delay model* allows gates and wires to have arbitrary delays. With the zero delay model, since all gates change with the clock, the number of synchronizations needed is small. On the other hand,

the unit and variable delay models produce lots of synchronizations between clocks, causing a huge amount of communication overhead.

To solve the synchronization problem, the design can be partitioned along flip-flop boundaries or according to the output cones so that signal changes propagate across boundaries only when the clock changes [6]. However, workload balancing will be more difficult given these constraints. Another way to reduce synchronization overhead while keeping the workload balanced is to sacrifice some timing accuracy. If the combinational part of the design is partitioned, then there will be signals propagating across partition boundaries between clocks. If we synchronize at constant intervals within clocks only, such as synchronizing once every 1/4 clock period, then we can reduce communication overhead and still propagate the signals. However, the timing becomes less accurate, implying a trade-off between communication overhead and timing accuracy. More synchronization points yield better timing accuracy but add more communication overhead. Since signals may propagate later in this scenario, it may cause false negatives on the output. Suppose, for instance, that the expected output is the value on the output when the pattern is simulated using the original timing model, then the output may be wrong when synchronization is reduced because of the delay in signal propagation across partitions. When this happens, the number of synchronization points between clocks should be increased.

Testing with scan chains has the advantage that during scan-in and scan-out, only signal propagations between scan registers are important, and these signals always change with the clock. Therefore we can synchronize at clock edges only and reduce communication overhead significantly. After all data have been shifted to the scan registers, simulation can be switched back to the variable delay model and run for one clock cycle. Simulation results using accurate timing information can thus be obtained. After that cycle, simulation can be switched back to clock-based synchronization for scan-out to reduce communication overhead. In this way, the required timing accuracy can be preserved while communication overhead is kept small.

To further reduce communication overhead, the partitions can also be built so that all the output ports of a partition are outputs of scan-chain flip-flops. In this way, synchronization can be reduced even in normal operating mode. However, gates or flip-flops may need to be duplicated with this technique, which introduces extra workload and may slow down the simulation. Therefore it should be treated an optional feature that can be turned on when necessary.

4.2 Workload Balancing

Since scan chain registers act as inputs and outputs of the combinational logic, they can be used as seeds to find good partitions – partitions with similar numbers of scan registers and combinational gates. In this way, each partition will have a similar number of primary inputs, primary outputs and gates. Since the input patterns are close to random, the workload of each partition will be balanced during simulation. The techniques of exploiting fixed nodes to find good partitions with minimal cuts have been discussed by Caldwell et al. [5]. By treating the scan registers as fixed nodes, such techniques can be used directly.

4.3 Partitioning Methodology

Based on our analysis above, we develop a partitioning methodology outlined in Figure 4. It is assumed that there are N partitions. The design is flattened to the library cell level before it is partitioned.

Most commercial ATPG tools produce testbenches according to their standard templates, which are often quite similar. Such templates include: (a) assignment of ports to scan chains, (b) force/release of registers in scan chains, and (c) assignment of patterns to ports. Instead of putting the testbench in a partition and

```
1 foreach partition P
2   Add 1/N scan registers to P;
3 Call min-cut partitioner with
  scan registers fixed;
4 Partition according to min-cut
  result;
```
Figure 4: Our partitioning methodology.

letting it communicate with other partitions, the testbench is partitioned to every partition and force/release calls are only performed for chains owned by the partition in order to reduce communication overhead.

5 Empirical Results

In this section we first describe how we configure our experiments and then present our parallel simulation results. In order to gain more insights into our methodology, we also study the impact of hypergraph partitioning and network latency on the performance of parallel simulation.

5.1 Experimental Setup

We conducted a series of experiments with SimCluster executing on different configurations including a Linux farm, Solaris SMP and a Solaris farm. The Linux farm is composed of Pentium-4 3.06GHz machines running Linux connected by Ethernet. The Solaris SMP machine is a four 1GHz UltraSPARC IIIi CPU machine running Solaris. The Solaris farm is composed of single 1GHz UltraSPARC IIIi CPU machines connected by Ethernet. The experiments simulate patterns generated by Synopsys TetraMax in serial mode and parallel mode, where serial mode is denoted by "(s)" in the benchmark and parallel mode is denoted by "(p)". In serial mode, patterns are shifted into and out of the scan chain cycle by cycle. In parallel mode, the patterns are directly forced into the scan chain. The target designs were a set of relatively large benchmark circuits which we summarize next. The balanced min-cut partitioner used in the experiments is hMetis [8].

Chip 1 is a graphics chip which contains approximately 12 million gates. It was partitioned manually due to memory limitations of the workstation used. B14, B17 and B22 are selected from the ITC'99 benchmark suite [23], and their sizes are on the order of 10K-100K gates, larger numbers indicate larger designs. B14 and B22 are subcircuits of the Viper processor, and B17 is a subcircuit of the 80386 processor. FFT is a Fast Fourier Transform circuit from Opencores [21], and its size is about 280K gates. All the benchmarks use the unit delay model and are created using Synopsys DFT flow. The original netlist of benchmark B14, B17 and B22 are flattened, while CHIP1 and FFT are hierarchical. The simulator used for CHIP1 is VCS, and the rest of the benchmarks use NC-Verilog. Since the process coordinator also simulates part of the design, for N partitions there will be $N+1$ processes.

5.2 Parallel Simulation Results

In order to show the effectiveness of our techniques, we ran several benchmarks using different configurations. The results of the experiments are summarized in Table 2. We can see that compilation was accelerated significantly for all benchmarks, which is a positive side-effect of partitioning the design. Since compilation time does not grow linearly with the size of the design, splitting the design to N partitions will often speed up compilation by more than N. This effect is especially obvious when library scan is used and

Benchmark	Parallel architecture	Number of partitions	Partitioner run time	Compile time			Simulation run time		
				Single process	Multi-process	Speed up	Single process	Multi-process	Speed up
B14(s)	Solaris SMP	2	45s	152s	10s	15.2X	51s	26s	2.0X
B14(p)	Solaris SMP	2	37s	168s	13s	12.9X	28s	41s	0.7X
B17(s)	Solaris SMP	2	758s	1300s	47s	27.6X	1048s	619s	1.7X
B17(p)	Solaris SMP	2	689s	1745s	50s	34.9X	426s	184s	2.3X
B22(s)	Solaris SMP	2	461s	1405s	33s	42.6X	447s	149s	3.0X
B22(p)	Solaris SMP	2	403s	1391s	36s	38.6X	279s	49s	5.7X
B22(s)	Solaris farm	2	461s	1061s	30s	35.4X	1608s	1275s	1.3X
B22(s)	Solaris farm	4	462s	1061s	28s	37.0X	1608s	978s	1.6X
B22(s)	Solaris farm	8	465s	1061s	26s	40.8X	1608s	1018s	1.6X
B22(p)	Solaris farm	2	403s	1066s	29s	36.8X	319s	155s	2.1X
B22(p)	Solaris farm	4	405s	1066s	28s	38.1X	319s	143s	2.2X
B22(p)	Solaris farm	8	406s	1066s	27s	39.5X	319s	124s	2.6X
FFT(p)	Solaris farm	2	675m	2515s	596s	4.2X	2655s	1121s	2.4X
FFT(p)	Solaris farm	4	697m	2515s	299s	8.4X	2655s	606s	4.4X
FFT(p)	Solaris farm	8	785m	2515s	218s	11.5X	2655s	328s	8.1X
CHIP1(s)	Linux farm	7	N/A	720m	60m	12.0X	470m	98m	4.8X

Table 2: Performance of distributed simulation.

the design is flattened, as in the case for the ITC benchmarks. For hierarchical designs, such as FFT, the compilation time reduction is closer to the number of partitions, as Figure 5 shows.

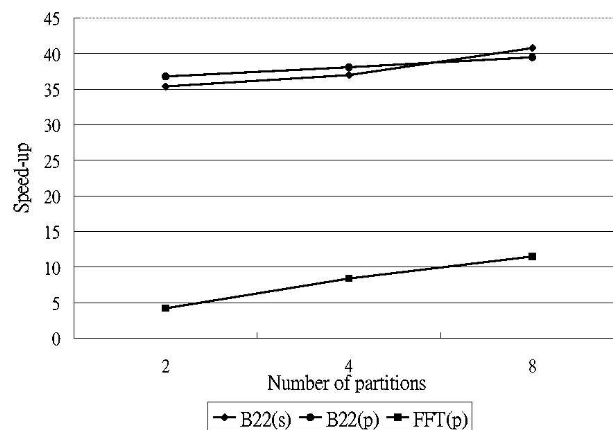

Figure 5: Compilation time speed-up.

The simulation speed-up shows that distributed simulation can reduce simulation time in all cases except b14(p), which is the smallest design and has the shortest single process run time. It can also be observed from the results that some speed-up factors are larger than the number of partitions, which may look strange at the first glance. The reason is that the simulator seems to take longer to read and initialize the design for simulation when the design gets larger. As a result, by splitting the design into smaller pieces, the initial setup time for simulation is also much shorter. This is another positive side-effect in addition to compilation time reduction. For longer simulation, this effect will diminish, and the speed-up will eventually be bounded

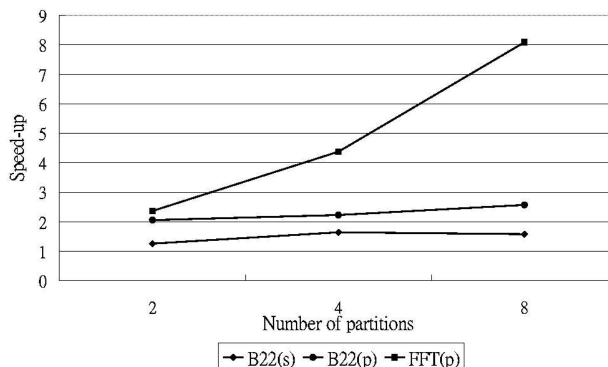

Figure 6: Simulation time speed-up.

by the number of partitions, as can be observed from B17, FFT and CHIP1.

The relationship between speed-up and the number of partitions of the three largest benchmarks is plotted in Figure 6. From the figure, we can see that the speed-ups of the B22 benchmarks saturate after two partitions, while the speed-up of the largest benchmark (FFT) continues to rise linearly. It is because communication overhead increases with number of partitions. If the workload in each partition is too small, communication overhead will dominate and speed-up will saturate. Since larger designs have higher workload, they can usually be distributed to more machines than smaller designs before the speed-up saturates. From the comparison of B22(p) and B22(s), it is observed that the speed-up of parallel mode is larger than serial mode. This is because in parallel mode, each cycle evaluates an ATPG pattern; while in serial mode, the ATPG pattern is shifted into the scan registers, and it behaves like a random pattern at each cycle. Since ATPG patterns have larger workload than random patterns, the workload-to-communication ratio is higher, which results in a better speed-up in parallel processing.

While the min-cut partitioner that we used scales well with the size of the design, the run time of our own partitioner does not grow linearly and becomes significant as the design gets larger. However, partitioning only needs to be done once and future changes in the patterns to be simulated do not require repartitioning. As a result, the partitioner's run time can be amortized when simulation of different sets of patterns is performed. Improving the efficiency of the partitioner is our next goal.

5.3 Impact of Hypergraph Partitioning

To show that balanced min-cut partitioning is critical to better performance, we compared it to a heuristic partitioner based on breadth-first search (BFS) which works as follows: (1) scan registers belonging to the same chain are added to each partition as seeds, and (2) other gates are added to the partitions using BFS with randomized selection of new unconnected gates. This heuristic should be more effective than random partitioning since random partitioning will partition scan registers belonging to the same scan chain to multiple partitions and cause a lot of communication overhead. To show the relationship between the quality of min-cut partitioning and the performance of parallel simulation, we also implemented a flat Fiduccia-Mattheyses (FM) [7] min-cut partitioner. In our experiments, "MLFM" uses hMetis [8] as the multi-level Fiduccia-Mattheyses partitioner and "FlatFM" uses our own flat FM partitioner. On average, partitions produced by flat FM have 3X greater cuts than those produced by multi-level FM. We ran the same benchmarks using our BFS heuristic, flat FM and multi-level FM partitioning, and the results are summarized in Table 3. A Solaris SMP machine is used for this experiment, and the number of partitions is two. From the results, we

Benchmark	Simulation run time(seconds)			
	None	BFS	FlatFM	MLFM
B14(p)	28	97	41	41
B17(p)	426	489	188	184
B22(p)	279	89	49	49
Average(speed-up)	1X	0.45X	2.9X	2.9X

Table 3: Comparison of speed-up using BFS heuristic and min-cut partitioning. None means the circuit is not partitioned.

can see that min-cut partitioning improves the performance of parallel simulation significantly – distributed simulation using min-cut partitioning runs 6.4X faster than heuristic partitioning. The comparison between flat FM and multi-level FM shows that while one can significantly improve upon simple BFS partitioning heuristics using a min-cut algorithm, the difference between the flat FM algorithm and its multi-level variant (which produces much better partitions much faster) is marginal in the overall simulation performance. This phenomenon can be explained by the following equation, which describes the time it takes to send a packet of data to a parallel process responsible for another partition: [1]

$$T_{packet} = T_{setup} + N \times T_{byte} \quad (1)$$

In the equation, T_{packet} is the total time to send a packet, T_{setup} is the time to setup the packet until the first byte of data can be sent. N is number of bytes to be sent, and T_{byte} is time to send an extra byte in the packet. Since in general, T_{setup} is much larger than T_{byte}, reducing N from 2000 to 1000 may be helpful, but reducing N from 200 to 100 may only reduce T_{packet} by a small fraction. Therefore flat FM should have similar performance compared with multi-level FM. As a result, while we only compared flat FM to multi-level FM in the case of two partitions, we expect that recursive bisection with flat FM partitioning will perform just as well as multi-way multi-level FM in multiple partitions.

5.4 Impact of Network Latency

Network latency is represented by T_{setup} in Equation (1). In order to gain more insight on Equation (1), we conducted an additional experiment using Gigabit Ethernet and Infiniband [22] with their latency roughly equal to 100μs and 10μs, respectively. In this experiment we use the Jazz DSP Platform from Improv Systems [24], which contains 15K flip-flops and 300K gates. This design was bi-partitioned by our min-cut partitioner. Parallel ATPG patterns were used as the test vectors, and the results are summarized in Table 4. In the table, "Packet count" is the number of packets sent through the network during simulation. "All events" means the simulation is synchronized whenever there is an event, "Every 500ps" means the simulation is synchronized every 500ps simulation timesteps, and "Every 5ns" means the simulation is synchronized every 5ns simulation timesteps. More synchronization points provide better timing accuracy but reduce the speed-up. Since total $N \times T_{byte}$ remains constant among all the simulations, the difference in speed-up is due to different amount of total T_{setup}. Comparison between different networks shows that reducing T_{setup} improves the performance of parallel processing significantly, especially when a large number of synchronizations is required. This can be observed by comparing the improvements in different synchronization modes: in "All", 6.5X performance improvement is observed; while only 1.3X improvement is

[1]While shared memory can significantly reduce this delay, it cannot support a computer farm, and so was not implemented at this time.

Sync. mode	Packet count (million)	Simulation speed-up		Improve-ment
		Ethernet	Infitiband	
All events	273	0.08X	0.52X	6.5X
Every 500ps	15.5	0.74X	2.12X	2.86X
Every 5ns	2.5	2.12X	2.75X	1.3X

Table 4: Comparison of speed-up using Ethernet and Infiniband with different number of synchronizations.

achieved in “5ns”. This experiment also shows that future improvements in reducing network latency may substantially increase the performance of parallel simulation.

6 Conclusions

In this paper, we revisited distributed parallel simulation of logic circuits. Unlike previous research, we focused on the effects of application-specific patterns in test vectors to the performance of parallel simulation instead of the effects of partitioning. By analyzing these patterns, we have successfully identified an application that appears to be well suited to parallel simulation – simulation of post-manufacture test patterns for scan-based ICs, which has high and balanced circuit activity. We proposed a new methodology to partition the design in order exploit these characteristics to balance workload and reduce communication overhead. While random partitioning performs almost as well as heuristic partitioning in functional tests, we found that it is not the case for manufacturing tests. Therefore we integrated a min-cut partitioner into our partitioning flow, and our partitioning methodology outperforms the use of a BFS heuristic with a 2.45X speed-up. When studying the impact of different techniques for min-cut partitioning on the performance of parallel processing, we have shown that compared to the flat Fiduccia-Mattheyses (FM) algorithm, multi-level FM brings only marginal improvements and is not worth the implementation effort. We also discussed the importance of reducing synchronization during simulation and suggested several ways to achieve it. Our experiments show that the proposed techniques can significantly accelerate the simulation of manufacturing test patterns and reduce simulation run time. As multi-core processors get popular and interprocessor communication becomes faster, such parallel simulation techniques will undoubtedly become more beneficial.

7 Acknowledgments

The authors want to thank Chris Browy (Avery Design Systems) and Mark Indovina (Improv Systems, Inc.) for their comments and suggestions.

References

[1] M. L. Bailey, J. V. Briner, Jr., and R. D. Chamberlain, “Parallel Logic Simulation of VLSI Systems”, *ACM Computing Surveys*, 1994, pp. 255-294.

[2] G. P. Balboni, G. P. Cabodi, S. Gai; D. Sismondi, and M. S. Reorda, “A Parallel System for Test Pattern Generation”, *Proc. of the Third IEEE Symposium on Parallel and Distributed Processing*, Dec. 1991, pp. 708-715.

[3] J. V. Briner, “Parallel Mixed-Level Simulation of Digital Circuits Using Virtual Time”, Ph. D Thesis, Duke University, 1990.

[4] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing*, Kluwer, Boston, 2000.

[5] A. E. Caldwell, A. B. Kahng and I. L. Markov, “Improved Algorithms for Hypergraph Bipartitioning”, *Proc. of the Asia and South Pacific Design Automation Conference*, Jan. 2000, pp. 661-666.

[6] K. H. Chang, W. T. Tu, H. W. Wang, Y. J. Yeh, and S. Y. Kuo, “Techniques to Reduce Synchronization in Distributed Parallel Logic Simulation”, *Proc. of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, Nov. 2004.

[7] C. M. Fiduccia and R. M. Mattheyses, “A Linear-Time Heuristic For Improving Network Partitions”, *Proc. of the ACM/IEEE Design Automation Conference*, 1982, pp. 171-181.

[8] G. Karypis and V. Kumar, “Multilevel k-way Hypergraph Partitioning”, *Proc. of Design Automation Conference*, Jun. 1999, pp. 343-348.

[9] R. H. Klenke, R. D. Williams, J. H. Aylor, “Parallel-Processing Techniques for Automatic Test Pattern Generation”, *IEEE Computer*, volume: 25, Issue: 1, Jan. 1992, pp. 71-84.

[10] M. Kormicki, A. Mahmood and B. S. Carlson, “Parallel Logic Simulation on a Network of Workstations Using a Parallel Virtual Machine”, *ACM Trans. on Design Automation of Electronic Systems*, Apr. 1997, pp. 123-134.

[11] Y. H. Levendel, P. R. Menon, and S. H. Patel, “Special-Purpose Computer for Logic Simulation Using Distributed Processing”, *Bell Syst. Tech. J. 61*, 10(1982), pp. 2873-2909.

[12] L. Li, H. Huang and C. Tropper, “DVS: An Object-Oriented Framework for Distributed Verilog Simulation”, *Proc. of the Workshop on Parallel and Distributed simulation*, 2003, pp. 173-180.

[13] N. Manjikian, and W. M. Loucks, “High Performance Parallel Logic Simulation on a Network of Workstations”, *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, May 1993, pp. 76-84.

[14] H. Nakahara, T. Sasao, and M. Matsuura, “A Fast Logic Simulator Using a Look Up Table Cascade Emulator”, *Proc. of The 11th Asia and South Pacific Design Automation Conference*, 2006, pp. 466-472.

[15] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham, “VLSI Logic and Fault Simulation on General-Purpose Parallel Computers”, *IEEE Trans. Computer Aided Design, Integrated Circuits and Systems, 12*, 3(1993), pp. 446-460.

[16] D. M. Nicol and P. R. Reynolds, Jr., “A Statistical Approach to Dynamic Partitioning”, *Proc. of the SCS Multiconference on Distributed Simulation*, San Diego, CA, 1992, pp. 139-146.

[17] S. P. Smith, B. Underwood, and M. R. Mercer, “An Analysis of Several Approaches to Circuit Partitioning for Parallel Logic Simulation”, *Proc. of the 1987 International Conference on Computer Design*, New York, Oct. 1987, pp. 664-667.

[18] S. P. Smith, B. Underwood and J. Newman, "An Analysis of Parallel Logic Simulation on Several Architectures", *Proc. of the 1988 International Conference on Parallel Processing*, Aug. 1988, pp. 65-68.

[19] S. Wang, S. K. Gupta, "ATPG for heat dissipation minimization during test application", *IEEE Tran. on Computers*, Volume 47, Issue 2, Feb. 1998, pp. 256-262.

[20] Avery Design Systems, "VCK/Simlib User's Guide", Rev. 1.1.0, May 2002.

[21] CF FFT Project, http://www.opencores.org/

[22] http://www.infinicon.com/

[23] ITC'99 Benchmarks(2nd release), http://www.cad.polito.it/tools/itc99.html

[24] Improv Systems: Jazz/DSP Core, http://www.improvsys.com/

[25] Verilog Simulator Benchmarks, http://www.veripool.com/verilog_sim_benchmarks.html