Efficient Simulation Based Verification by Reordering

Chao Yan
Department of Computer Science
University of British Columbia
Vancouver, BC, V6T1Z4, Canada
chaoyan@cs.ubc.ca

Kevin Jones
Green Plug, Bishop Ranch 2
2694 Bishop Drive, Suite 209
San Ramon, CA 94583, USA
drkdjones@gmail.com

***Abstract*—With the increasing complexity of systems, the current simulation based circuit verification used in industry are becoming more expensive while providing low coverage. The paper presents a systematic way to reduce the verification time by optimizing the execution order of test cases. Compared with the default order maintained by engineers, the optimal order can achieve a high coverage in a short time as it guarantees running the important test case first.**

We developed both *offline* and *online* algorithms to find the optimal order that maximizes the ratio of coverage to verification time. The offline algorithms analyzes the simulation data and compute an efficient order for later executions. Three algorithms are provided to find the optimal, near-optimal and super-optimal solutions respectively. To find a better order without any precomputed data, an online algorithm is developed. The algorithms predicts the coverage and running time of a new test case by taking advantage of special properties of test cases.

We applied our algorithms to the verification of XIO devices from Rambus Inc. It is shown that the order produced by the offline based greedy algorithm saves up to 85% of verification time. And the result from the online algorithm saves about 50% time on average.

I. Introduction

According to the Moore's law, the number of transistors on a single chip doubles every 18 months. But the circuit design and verification productivity increase at a much lower rate, which causes a gap between physical implementation and circuit design and verification. During the past decade, verification time has taken up an increasing amount of the design cycle. Thus, significant research effort has gone into the modeling and verification of digital circuit [1] since electronic devices are largely composed of digital circuits. However, verifying analog and mixed signal (AMS) circuits, which play a crucial role when interfacing with inherently analog environment to digital components, is still an open problem.

Formal verification and simulation are two main approaches to verifying that a system satisfies a property or specification. Even though formal methods, like model checking [2], [3], can provide higher reliability and have gained great success in digital circuit verification, in practice simulation based verification is still used in industry. Over the years, design methodology has relied heavily on simulation to verify the correctness of AMS circuits. However, the increased circuit complexity makes each single simulation notoriously long. Process variation also increases the difficulty of AMS circuit verification. Although Monte Carlo simulation [4] is used for growing ranges of process variations and environmental conditions, a large testbench is required to cover all kinds of corner cases. As a result, the simulation takes several weeks or even months for industrial scale AMS circuits.

This work was done when both authors were at Rambus Inc., Los Altos, California

The problem is compounded by the ad-hoc nature of today's simulation based verification. In a typical circuit design process, the system level specification is set by the system architects based on customers' requirements, industry standards, etc. After the architectural decisions have been made, designers begin to design individual subsystems and run large numbers of simulations to check the implementation. Right to the end of development, more test cases to verify the complete system are added to the testbench. Because the collection of test cases are developed by many engineers during a long period, redundancies are inevitable in such a testbench.

Techniques to analyze and optimize the testbench and to improve the performance of verification are very useful for industrial circuit design. We noticed that the efficiency of verification highly depends on the execution order of test cases. The currently used order is maintained by engineers manually and usually is not optimal. This paper proposes a method to reduce verification time and increase simulation coverage by reordering the execution order automatically. Several algorithms are developed to compute the optimal order, which puts the most important test cases at the beginning and redundant ones at the end. Experimental results from real industrial devices demonstrate the efficiency of our solution.

The paper is organized as follows. The next section describes the problems and our mathematical model. Section III presents our offline and online algorithms to find the optimal execution order. Section IV shows how to apply our algorithms to real circuits and the experimental results. Finally, our conclusion and future work are presented.

II. Problem and Definitions

This section introduces the necessary basic notions and definitions to build a mathematical model for the verification problem.

In this paper, the *specification* $F = \{f_1, \cdots, f_m\}$ defines all *functions* f to be implemented and verified. And the *testbench* $J = \{\hat{\jmath}_1, \cdots, \hat{\jmath}_n\}$ consists of a set of *test case* $\hat{\jmath}$, which is usually a standalone simulation file. The *running time* t of a test case refers to the time to complete the verification and T is the total verification time.

Usually, one test case checks several functions and each function is verified by several test cases. The relationship between test cases and functions is represented as a *coverage matrix* M. The element of the matrix indicates whether a test case verifies a function or not , as defined in Equation 1.

$$M_{m,n} = \begin{cases} 0 & \hat{\jmath}_n \text{ does not check } f_m \\ p & \hat{\jmath}_n \text{ covers a part of } f_m \\ 1 & \hat{\jmath}_n \text{ fully covers } f_m \end{cases} \tag{1}$$

We call the column of coverage matrix, denoted as V, the *coverage vector* of the test case. If two coverage vectors V_i and V_j have non-zero elements on the same position, there might be *redundancy* between test case $\hat{\jmath}_i$ and test case $\hat{\jmath}_j$.

A vector W is used to represent the weights of functions in the specification if they are not equally weighted. Then the *coverage* C of a test case is defined as

$$C = \langle W \cdot V \rangle,$$

where $\langle a \cdot b \rangle$ is the inner product of vectors a and b.

The *set coverage vector* for a set of test cases $\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}$ is defined as [1]

$$V_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} = \min(1, \sum_{i=1}^{k} V_i).$$

The corresponding *set coverage* is defined as

$$C_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} = \left\langle W \cdot V_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} \right\rangle.$$

And the *coverage increment* for test case j_i, as defined in Equation 2, is the difference of coverages before and after adding j_i to the set $\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}$.

$$\Delta C^{\hat{\jmath}_i}_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} = C_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k, \hat{\jmath}_i\}} - C_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} \tag{2}$$

It is noticed that the coverage increment of a test case depends not only on its coverage vector but also the test cases executed before. Therefore, the execution order is crucial to reduce the redundancy and improve efficiency of verification. As it is expected to gain high coverage as soon as possible, it is better to run the most important test case at the beginning. In this paper, the verification *efficiency* E is defined as the ratio of coverage to total running time

$$E_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} = \frac{C_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}}}{T_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}}}.$$

And the *importance* r of a test case refers to the ratio of coverage increment to running time

$$r^{\hat{\jmath}_i}_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}} = \frac{\Delta C^{\hat{\jmath}_i}_{\{\hat{\jmath}_1, \cdots, \hat{\jmath}_k\}}}{t_i} \tag{3}$$

With these definitions, the problem of improving verification performance is translated to an optimization problem: how to find an order of J to maximize the verification efficiency?

III. Algorithm

We developed several algorithms to solve the above problem. First, three offline algorithms are presented in Section III-A to reduce the verification time for the case when the testbench is executed multiple times. Then, an online algorithm which tries to guess a good execution order to speedup the verification without any precomputed information is described in Section III-B.

A. Offline Algorithm

In the typical design process, circuit design and verification interact with each other. The design is modified if it fails to pass the verification. Even though the modification might be minor, the testbench should be run and checked again. Thus, the testbench is often executed many times during the whole development period.

Noticing that the running time and coverage vectors are similar during different executions, we developed offline algorithms to speedup the verification. The algorithms first run all test case in the testbench $J = \{\hat{\jmath}_1, \cdots, \hat{\jmath}_n\}$, and then collect the simulation data including the running time $\{t_1, \cdots, t_n\}$, the coverage matrix M and also the weight vector $W = \{w_1, \cdots, w_m\}$. With this information, the algorithms find an optimal execution order to be used later.

The algorithms also solve two problems which are especially important when there is not enough time to complete the testbench. The first is to find a subset of J to achieve maximum coverage within given time T, and the other is to find a subset of J to achieve given coverage C within minimum time. We present three algorithms with different complexities in the following sections.

1) Optimization Problem: The first algorithm solves the problems by converting them to integer linear programming [5] (ILP) problems.

The problem of maximizing function coverage within given time T is converted to an ILP problem as:

$$\begin{aligned}
& max && \sum_{j=1}^{m} h_j \cdot w_j && \text{(4a)} \\
T & \geq && \sum_{i=1}^{n} x_i \cdot t_i && \text{(4b)} \\
h_j & \leq && \sum_{i=1}^{n} x_i \cdot M_{j,i} \quad (j = 1, \cdots, m) && \text{(4c)} \\
h_j & \geq && 0 \quad (j = 1, \cdots, m) && \text{(4d)} \\
h_j & \leq && 1 \quad (j = 1, \cdots, m) && \text{(4e)} \\
x_i & \in && [0, 1] \quad (i = 1, \cdots, n), && \text{(4f)}
\end{aligned}$$

[1]Here, we over estimate the coverage by assuming there is no redundancy between test cases. The estimation error is usually small because it is common that most elements of coverage matrix M are either 0s or 1s in a typical project. The min operation is used to ensure the total coverage of any function does not exceed 1.

where x_i is a boolean variable that indicates whether test case $\hat{j}_i$ is in the subset or not, w_j is the weight of function f_j, h_j is the total coverage of function f_j, n is the number of test cases and m is the number of functions. The linear inequality from Equation 4b adds the constraint that the total verification time is not longer than T. Equation 4c,4d,4e constraints the value of h_j. The optimization goal is to maximize the total coverage as shown in Equation 4a.

Similarly, the problem of minimizing running time with given coverage C is also converted to a 0-1 ILP problem:

$$\begin{array}{rcl} & min & \sum_{i=1}^{n} t_i \cdot x_i \\ C & \leq & \sum_{j=1}^{m} h_j \cdot w_j \\ h_j & \leq & \sum_{i=1}^{n} x_i \cdot M_{j,i} \quad (j = 1, \cdots, m) \\ h_j & \geq & 0 \quad (j = 1, \cdots, m) \\ h_j & \leq & 1 \quad (j = 1, \cdots, m) \\ x_i & \in & [0,1] \quad (i = 1, \cdots, n). \end{array}$$

The differences is that Equation 4b is replaced with the constraint for total coverage, and the optimization goal is to minimize the verification time.

2) Greedy Algorithm: However, the general ILP is a NP hard problem [5]. Therefore, we developed another efficient approximation algorithm to compute an near optimal solution. The algorithm is a greedy algorithm. It always finds and executes the most important test case at each step, as shown in Algorithm 1. Line 4 computes the importance of a test case as defined in Equation 3.

Algorithm 1: Greedy Algorithm

```
Input: A testbench $J = \{\hat{j}_1, \cdots, \hat{j}_n\}$, coverage matrix
       $M$, and running time $T = \{t_1, \cdots, t_n\}$
Output: The execution order
1 for $i = 1, \ldots, n$ do
2     $\Delta R = 0$;
3     for $k = i, \ldots, n$ do
4         $r = \frac{\Delta C^{\hat{j}_k}_{\{\hat{j}_1, \ldots, \hat{j}_{i-1}\}}}{t_k}$;
5         if $r > \Delta R$ then
6             $\Delta R = r$;
7             $m = k$;
8     add $m$ to the end of execution order ;
```

From the experimental results in section IV, it is shown that the result from this greedy algorithm is much better than the default order. However, we want to know how big is the gap between the optimal result and the approximated one from greedy algorithm. Therefore, we developed the third algorithm to find a super-optimal solution.

3) Super-optimal Solution: Although there is no efficient algorithm to solve general ILP problem, there are many efficient ways, like Simplex [6] algorithm, to solve linear programming (LP) problem. Therefore, the third algorithm converts the ILP problems to LP problems by relaxing the integer constraint in Equation 4f. The solution of relaxed LP problem is a super-optimal solution which might be better then the optimal one when it violates the constraints of original ILP problems. The relaxed LP problems are:

$$\begin{array}{rcl} & max & \sum_{j=1}^{m} h_j \cdot w_j \\ T & \geq & \sum_{i=1}^{n} x_i \cdot t_i \\ h_j & \leq & \sum_{i=1}^{n} x_i \cdot M_{j,i} \quad (j = 1, \cdots, m) \\ h_j & \geq & 0 \quad (j = 1, \cdots, m) \\ h_j & \leq & 1 \quad (j = 1, \cdots, m) \\ x_i & \geq & 0 \quad (i = 1, \cdots, n) \\ x_i & \leq & 1 \quad (i = 1, \cdots, n) \end{array}$$

and

$$\begin{array}{rcl} & min & \sum_{i=1}^{n} t_i \cdot x_i \\ C & \leq & \sum_{j=1}^{m} h_j \cdot w_j \\ h_j & \leq & \sum_{i=1}^{n} x_i \cdot M_{j,i} \quad (j = 1, \cdots, m) \\ h_j & \geq & 0 \quad (j = 1, \cdots, m) \\ h_j & \leq & 1 \quad (j = 1, \cdots, m) \\ x_i & \geq & 0 \quad (i = 1, \cdots, n) \\ x_i & \leq & 1 \quad (i = 1, \cdots, n), \end{array}$$

which allow x_i to be any value between 0 and 1.

With these three algorithms, we first compute the near optimal order using the greedy algorithm and compare the result with the super-optimal solution by solving relaxed LPs. If the gap is small enough, the solution from greedy algorithm is close to the optimal one, thus it is not necessary to solve the expensive ILP problems. Otherwise, the optimal solution is computed by solving the ILP problems using commercial tools, like CPLEX [7]. Because the size of our problem is not huge, the computation is still efficient, especially for some special form of ILP problems. From our experimental result, the greedy algorithm works well for most of the times.

B. Online Algorithm

The offline algorithms described above compute an optimal order based on the previous simulation data. However, it is still necessary to speedup the first execution of the testbench because a single run of all test cases is also very expensive. Therefore, an online algorithm is developed to find an order better than the default one without the knowledge of coverage matrix and running time.

Firstly, the algorithm predicts the running time t_k as the length of its coverage vector $|V_k|$. The assumption that the running time t_k is proportional to $|V_k|$ is reasonable because it usually costs more time to check more functions. It is also supported by the experimental results. We replace the t_k term with $|V_k|$ in line 4 of the greedy algorithm and find the result changes slightly.

To predict the coverage vector of a new test case, we analyzed the special structure of test cases used in the project. All test cases in the testbench consist of a *configuration* and a *feature*. The configuration usually defines the environment, modes of operation, and parameters of the device. A feature is usually a basic operation of the device, which may cover several different functions depending on the configuration. The testbench contains all possible combinations of configurations and features. We use $\hat{\jmath}_{c_i,f_j}$ to represent a test case which uses configuration c_i and feature f_j, and use V_{c_i,f_j} to represent its corresponding coverage vector.

The *similarity* of two test cases $\hat{\jmath}_i, \hat{\jmath}_j$ is defined as the angle between their coverage vectors V_i, V_j:

$$sim(\hat{\jmath}_i, \hat{\jmath}_j) = \frac{\langle V_i \cdot V_j \rangle}{|V_i| \cdot |V_j|}$$

The similarity of test cases with same configuration or feature is usually large. For example, two test cases with the same feature usually have similar running path and thus cover similar functions. Therefore, we also compute the similarity of features and configures. The *configuration similarity* of two configurations c_i, c_j is defined as

$$csim(c_i, c_j) = \prod_{k=1}^{nf} sim(\hat{\jmath}_{c_i,f_k}, \hat{\jmath}_{c_j,f_k})^{\frac{1}{nf}} \quad (5)$$

where nf is the number of features. And the *feature similarity* of two features f_i, f_j is defined as

$$fsim(f_i, f_j) = \prod_{k=1}^{nc} sim(\hat{\jmath}_{c_k,f_i}, \hat{\jmath}_{c_k,f_j})^{\frac{1}{nc}} \quad (6)$$

where nc is the number of configurations. It is found that the coverage similarity is approximately the product of its configuration similarity and the feature similarity:

$$sim(\hat{\jmath}_{c_1,f_1}, \hat{\jmath}_{c_2,f_2}) \approx csim(c_1, c_2) \cdot fsim(f_1, f_2)$$

The assumption is verified by the experimental result. The average error of using this approximation is less than 0.08% and the maximum error is less than 4.25%. Thus, the product of configuration similarity and feature similarity is a good approximation of the coverage similarity.

With the predication of running time and similarity between coverage vectors, the importance r as defined in Equation 3 is approximated by:

$$\begin{aligned} r^{\hat{\jmath}_i}_{\{\hat{\jmath}_1,\cdots,\hat{\jmath}_k\}} &\approx 1 - \sum_{l=1}^{k} sim(\hat{\jmath}_l, \hat{\jmath}_i) \cdot \Delta C^{\hat{\jmath}_l}_{\{\hat{\jmath}_1,\cdots,\hat{\jmath}_{l-1}\}} \quad (7) \\ &\approx 1 - \sum_{l=1}^{k} csim(c_{\hat{\jmath}_l}, c_{\hat{\jmath}_i}) \cdot fsim(f_{\hat{\jmath}_l}, f_{\hat{\jmath}_i}) \cdot \Delta C^{\hat{\jmath}_l}_{\{\cdots\}} \end{aligned}$$

The intuition behind the equation is that if the coverage vectors V_i, V_l are very similar then test case $\hat{\jmath}_i$ is less important because lots of functions in test case $\hat{\jmath}_i$ have already been checked by test case $\hat{\jmath}_l$.

Algorithm 2 shows the implementation of the online algorithm. It is similar with the greedy algorithm, but at each step it finds the test case with minimum similarity with other test cases already executed. The importance of a test case is predicted using Equation 7 in lines 5-12. After the execution of the new test case, its coverage vector is used to update the configuration similarity and feature similarity according to Equation 5, 6.

Algorithm 2: The Online Algorithm

```
Input: A testbench $J = \{\hat{\jmath}_1, \cdots, \hat{\jmath}_n\}$
Output: The order of test cases to run
1  $csim = 1, fsim = 1$;
2  for $i = 1, \ldots, n$ do
3      $\Delta R = \infty$;
4      for $k = i, \ldots, n$ do
5          $r = 0$ ;
6          for $l = 1, \ldots, i-1$ do
7              $w = \Delta C^{\hat{\jmath}_l}_{\{\hat{\jmath}_1,\ldots,\hat{\jmath}_{l-1}\}}$;
8              $sim = csim(c_{\hat{\jmath}_k}, c_{\hat{\jmath}_l}) \cdot fsim(f_{\hat{\jmath}_k}, f_{\hat{\jmath}_l})$;
9              $r = r + w \cdot sim$;
10         if $r < \Delta R$ then
11             $\Delta R = r$;
12             $m = k$;
13     execute $\hat{\jmath}_m$ and collect the coverage and running time
       data;
14     update $csim$ and $fsim$ as shown in Equation 5 and
       Equation 6;
```

IV. Experimental Result

We applied our algorithms to the verification of two XIO devices [8] developed by Rambus Inc. The XIO cell is a high-performance, low-latency controller interface to XDR DRAM memory devices.

The verification platform used in this project is *specman*. Figure 1 shows its architecture. *DUT* is the device to be verified, which is the XIO device in this paper. The *transaction generator* generates transactions based on the test constraints and drives inputs to the DUT. The *output checker* monitors the output signals from DUT and compares them with the correct results from *predictor*. The coverage data and other information are collected during the simulation by the *coverage* block.

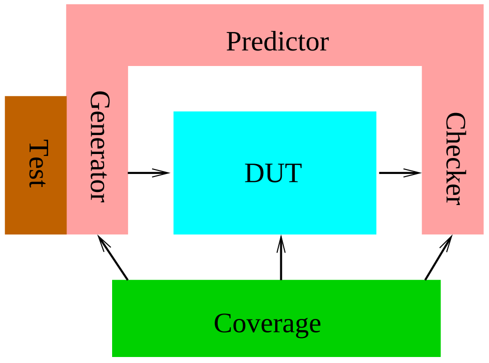

Fig. 1. The Verification Framework

The XIO devices are implemented in the *verilog* language and simulated by Cadence's *nc-sim* simulator. The test cases are coded with the e language. And the *vManager* tool is used to collect coverage data required by our algorithms. Our algorithms are implemented in the Matlab platform which has the advantage of fast prototyping and powerful visualization.

The result of offline algorithms are described first, with the comparison with the result from vManager. Then several methods are used for the online problems and their results are compared.

A. Result of Offline Algorithms

We apply both the greedy algorithm and relaxed LP based method to the Toshiba XIO device. Figure 2 shows result, where the horizontal axis is for the verification time and the vertical axis indicates the coverage.

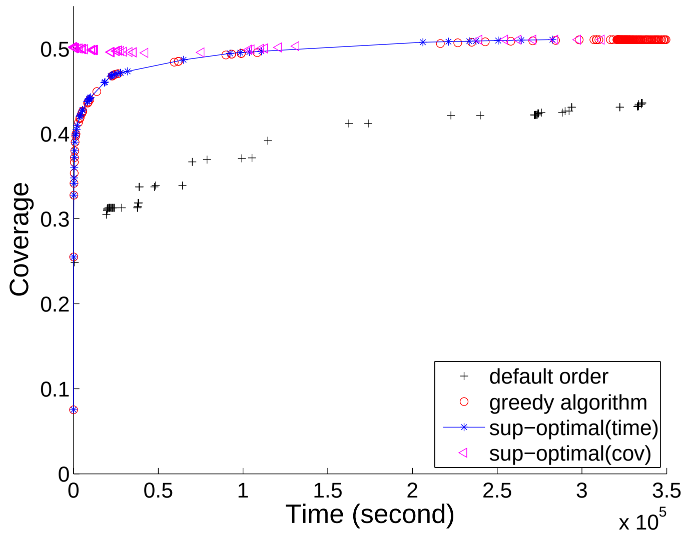

Fig. 2. Result of Offline Algorithms

In the figure, we noticed that the result from greedy algorithm is much better than the default order and is quite close to the super-optimal order by solving relaxed LP. Although there is a large gap between approximated solution and the super-optimal one at the beginning, we do not think the super-optimal solution is feasible because the integer constraint of Equation 4f is seriously violated. Therefore, there is little benefit to solving the expensive ILP to compute the exact optimal result. From the greedy algorithm's result, we can see that 169 over 222 test cases are redundant thus can be removed without reducing coverage, or more than 85% time can be saved using the execution order from greedy algorithm. It should be mentioned that the coverage increases to 40% rapidly within 2% of total verification time, which is very useful when there is not enough time to run all test cases to achieve a coverage of 50%.

vManager also provides a *ranking* function to rank the importance of all test cases. We compare the results from vManager and our greedy algorithm. As shown in Figure 3, it is obviously that the results from vManager and greedy algorithm are almost the same. Thus we believe vManager also uses a similar greedy algorithm.

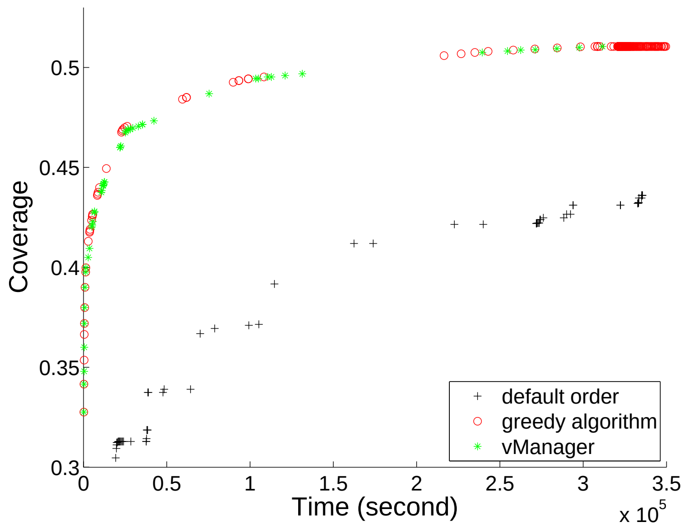

Fig. 3. Result from vManager

B. Result of Online Algorithms

Then the online algorithm is applied to a similar BE-XIO device. To check the efficiency of our algorithm, several other orders are also considered. The *default order* maintained by engineers lists all test cases mainly by its creation time. The *column first order* puts all test cases with the same configuration together, and the *row first order* puts all test cases with the same feature together. The *random order* select the next test case to execute randomly. The results for these execution order are shown in Figure 4.

Obviously, the result for the default order, shown as the black curve in the plot, is the worst one. The column first order and row first order are a little better. And the column first order is usually better than the row first order, which indicates that the similarity of test cases with same feature is usually larger. The blue curve in the plot shows the average result for

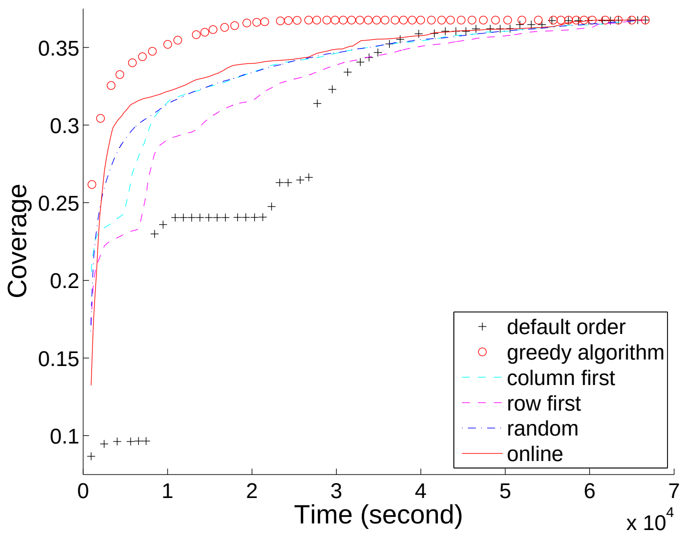

Fig. 4. Result of Online Algorithms

random order, which is slightly better than column first order at the beginning. But there is still big gap between random order and the near optimal order from greedy algorithm.

The online algorithm as shown in algorithm 2 produces better result than all other pre-defined orders. However, its result is close to the result from random order at the beginning when the algorithm does not have enough data to predict the importance of a test case accurately. Similarly, the results are similar at the end which can be explained by the large accumulated estimation error in Equation 7.

V. Conclusion and Future Work

In summary, we improved the performance of simulation based verification by reordering the test cases. We developed both offline algorithms and an online algorithm. The greedy algorithm produces a close to optimal execution order and the online algorithm makes the first execution of the testbench more efficient by predicting the coverage vector and running time of a test case. The results from two XIO devices demonstrate that our algorithms can save large percentage of time for practical verification.

The methods presented in this paper are general and can be applied to other AMS circuits as well. First, the offline algorithms can be applied to any verification problem given the precomputed coverage information. Second, the online algorithm can be extended to support similar problems in other projects. Of course, different methods to predict coverage vectors are required for other testbenches with different structures.

Further improvement can be gained by applying similar algorithm to small granularity given it works for large granularity. The testbench is composed of test cases, similarly, a test case usually consists of several components. Optimizing within a test case would be more powerful and reduce more redundancy. However, the reordering method does not work directly for components because they are not independent with each other as are test cases. On the other hand, the coverage of a test case depends on both the order of components and their combinations. This also provides the opportunity to generate new test cases and increase coverage automatically by combining components from different user provided test cases. Of course, it is challenging to handle the dependency between components and generate functionally correct test cases.

The online algorithm can be also extended to small granularity. The components in a test case usually have multiple operation modes, for example, there are several different ways to initialize the XIO device. However, it is usually impossible to check all possible cases because the number of combinational cases is huge especially for large circuit. Therefore, only a small subset of typical cases are considered due to time constraints. Now the subset is usually selected manually or randomly which can not guarantee a good result. The online algorithm can be extended to solve the problem because it does not require any information of test cases and produces a close to optimal result automatically.

Acknowledgment

We would like to thank Mark Greenstreet for his advice on the mathematical model, and also thank Kathryn Mossawir, Tom Sheffler, John Hong and Victor Konrad for their help on this project.

References

[1] K. Thomas, *Introduction to Formal Hardware Verification*. Springer, 1999.

[2] E. M. Clarke and B.-H. Schlingloff, *Model checking*. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 2001.

[3] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[4] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[5] A. Schrijver, *Theory of linear and integer programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[6] N. Jorge and W. Stephen, *Numerical Optimization*. Springer, 2000.

[7] "Using the cplex linear optimizer," CPLEX Optimization Inc, Incline Village, NV, 1994.

[8] "Xdr io cell technical document." [Online]. Available: http://www.rambus.com/us/downloads/document_abstracts/products/dl_-0362_v0_71.html