# Mining Coverage Data for Test Set Coverage Efficiency

Monica Farkash, UT Austin
Bryan Hickerson, IBM Austin
Mike Behm, IBM Austin
Balavinayagam Samynathan, UT Austin

***Abstract*- We present a coverage data mining method of analyzing the efficiency of the verification process. We show how to use a mathematical model to change the test generation to reach higher coverage efficiency. We demonstrate a self-adaptive solution which focuses on hard-to-hit events with a 'time' dimension, increasing the likelihood of hitting these events by adjusting the test load, shifting the effort from a very expensive manual effort to the existing automatic process.**

**For IBM's PowerPC core verification our results show 12% reduction in hard-to-hit events (less than 2,000 hits for every 1,000,000 tests) and 13% reduction in never-seen events.**

## I. INTRODUCTION

Verification progress is often measured using coverage events, with an ultimate goal of 100% coverage. The process consists of automatic test generation driven by pre-defined test scenarios, test simulation and coverage measurement collection (figure 1). It is complemented by the manual targeting of the hard-to-hit coverage events (including the never-hit events). Hard-to-hit events are defined as less than 2,000 hits for a moving window of 1,000,000 tests in real time. Never-hit-events are defined as never being hit in the rolling window.

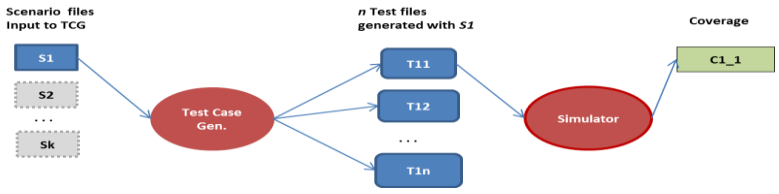


**Figure 1. The Verification process**

A more coverage efficient verification process would bring a change in the coverage landscape it generates which would exhibit a shift in the number of hits, increasing it among hard-to-hit events, and decreasing it among the easy-to-hit events. Process efficiency increase would result in the reduction not only of verification cost but also of time to market.

There are two aspects of increasing the verification process coverage efficiency: reducing the effort invested into targeting the hard-to-hit events, and also reducing the redundancy among the easy-to-hit events. While the latter is less identified as a problem, the cost of manually targeting hard-to-hit events is. For each such event the process consists of understanding the conditions required to generate the event, expressing them in terms recognized by the

test or the test generator, generating the test, simulating it, and reiterating the process with refinements until it succeeds in covering the event. Driving the hardware towards hitting the event often requires a tight control over test execution which is frequently unavailable.

## II. Prior Work

*Coverage Driven Verification* [1] [2] is the concept of using coverage as the main goal of the verification process. Considering that the process starts with a test run on the HW (hardware), *coverage directed* and *coverage-driven test generation* use coverage data to control the test generation to achieve coverage [3] [4] [5]. Graph-based test generators take decisions which test to generate based on coverage. These solutions focus on the coverage events and on targeting them directly if they are difficult to achieve randomly. A major hindrance to directly target coverage events is that the targeted event has to be expressed at the level of the test that is run, or the test generation, which is rarely possible. In our industrial example, most of the 150k coverage events, and especially those hard –to-hit events, do not satisfy this condition.

Data mining allows us to heuristically guide the verification towards achieving coverage. It does not attempt to generate tests which will definitely hit the event, but rather to generate tests with and increased likelihood of hitting the events without requiring direct controls over the simulation process or strong controls over the test generation.

## III. Our Approach

Instead of treating coverage redundancy on easy-to-hit events and lack of coverage on hard-to-hit events as two distinct problems, they are two aspects of the same problem: the fact that the existing automatic solution is not efficient. A more efficient process would change the coverage count distribution on the events as part of the already existing automatic process. The efficiency increase doesn't remove the need of specially handling some hard-to-hit coverage events but reduces considerably the number of such events.

This approach starts with the following observations:

1) Due to the random nature of the test generation, the coverage results gathered from a large amount of tests would show a data distribution which can be analyzed as a statistical value.

2) All tests generated with a given scenario have certain aspects in common, which reflect the functionality targeted by the scenario, and will be further mirrored in the coverage event distribution they generate while being simulated.

3) There is a correlation between the load (number of instructions) of the test and the number of cycles it runs. Due to the large variations in the number of cycles instructions take, depending on the instruction and the HW internal conditions, there is not a linear relation between the number of instructions and the number of cycles a test takes to simulate. For the purpose of this analysis we consider the number of instructions in the test as being the parameter we can control to influence the length in cycles the test takes.

4) We assume that for the purpose of this analysis both shorter and longer tests, for the amount of cycles they have in common, exhibit similar coverage events.

5) Due to the price paid for generating tests with a large number of instructions and the time spent running them, 30,000 cycles simulation runs are a reasonable upper limit for the industrial available verification environment used in this research to learn about scenarios and their coverage event distributions.

## IV. The Verification Process

The verification process overview was shown earlier in figure 1. An automatic test generator is used to generate pseudo-random tests. There are manually written scenario files, which are read by the generator, which contain the definition of what is required for the future test. The test conditions not defined in the scenario file are being chosen randomly (bias). One of the variables that feeds into the test generation is the size of the test. This controls the number of instructions eventually generated in the test, and through that, it influences the amount of traffic the test generates and the number of cycles it will run when simulated.

## V. Different Aspects Of Coverage Data

We use coverage data to measure the quality of our verification. We analyze the coverage as follows (data from IBM POWER processor verification process):

### *A. Coverage in Time*

Coverage in time provides an insight into when coverage happens. Coverage events represent important parts of HW functionality hence they extract information corresponding to the functionality that matters to the verification process. Figure 2 shows the evolution of coverage events during a test simulation. The *x* axis represents the simulation cycle and the *y* axis represents the number of coverage events hit in that particular cycle. There seems to be coverage generating test activity throughout the simulation run.

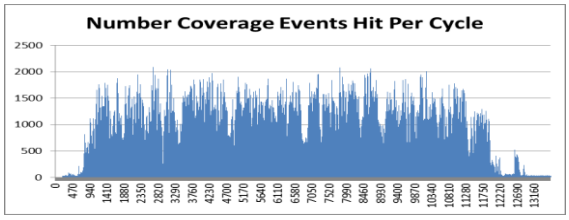


**Figure 2. Number of coverage events hit each cycle**

### *B. Covered: True/False*

The amount of hits for a given event in a given time doesn't distinguish between one test hitting the same event often or several tests exercising that area once. Hitting the same event in different tests could mean a larger variety of external conditions under which the event was achieved. We work with the assumption that, if repetitive hits in the same test are important, there would be coverage events which would reflect this notion. Given this, the coverage recording can be limited to a Boolean recording if the event was hit or not.

### *C. First Time per Test (FTPT)*

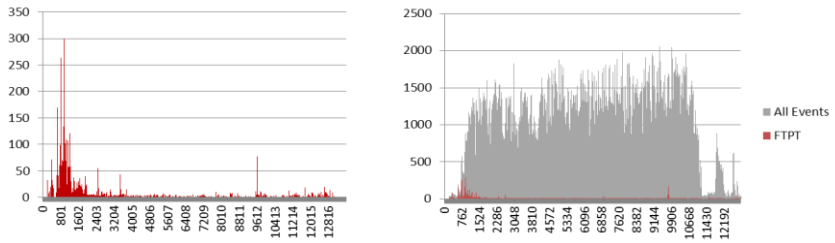


**Figure 3. FTPT events as small part of all events**

First Time Per Test (FTPT) coverage of an event is the simulation timestamp of when that particular event is reached for the first time while simulating a test. FTPT coverage excludes redundancy from the analysis. Figure 3 shows side by side the overall coverage of a test in time, and the FTPT coverage. If FTPT is the measure of efficiency, the first part of the simulation run provides a higher coverage efficiency than the rest.

### *D. Understanding the FTPT*

The FTPT coverage presents a pattern which can be described as *waves* of events. A test simulation example shows the first *wave* is easily distinguishable, with the highest points around cycle 1000. **Figure** presents automatically extracted *waves* based on Expectation Minimization [6] [7] [8]. For example, figure 4 (a) is showing how data for the first recognizable 3 waves is being fitted to the model and continues to be correct for subsequent wave shown in figure 4 (b).

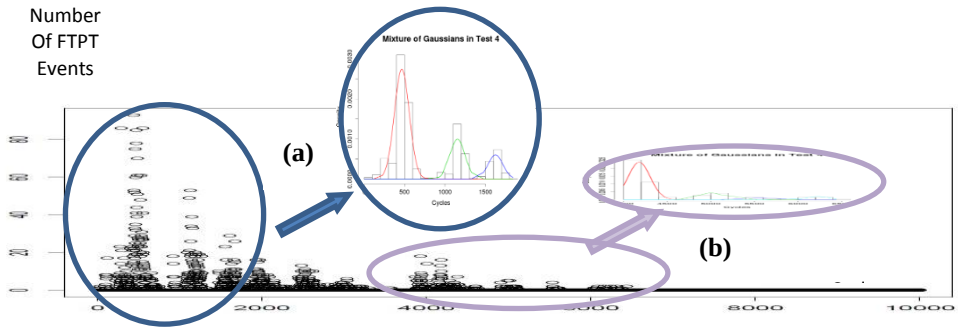


**Figure 4. Coverage FTPT as overlapping distributions**

The coverage accomplished by a test is not a linear function in time, in which the longer the test runs the higher the probability of hitting an event. The coverage behaves as a mixture of distributions. Each such distribution can be understood as a reflection of a functionality opened at a given time. For example, the first time a floating point instruction is being run, it will *open*, *exercise*, the design area dedicated to that type of instruction which results into such a wave. Model fitting identifies the FTPT waves as gamma distributions as shown in figure 5, which is consistent with the representation of the *cumulative distribution function* (CDF) of a Gamma distribution compared to the practical expectation of coverage achievement in time.

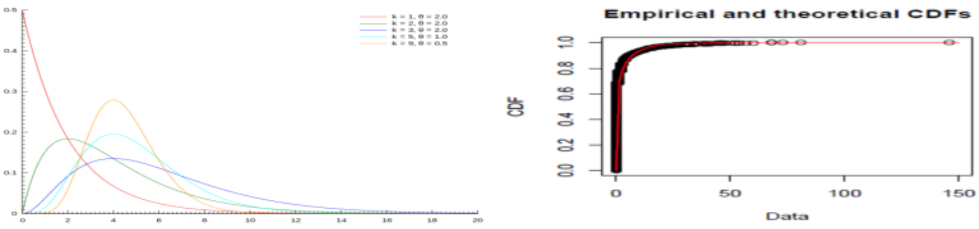


**Figure 5. Gamma distribution matching**

Mapping a *wave* to the opening of a new area of functionality brings a new way of looking at how coverage happens. The purpose of test generation is thus not to focus on generating instructions, but, as it is already intended

with the scenario files, to focus on generating coverage *waves*, each wave meaning the *opening* of an area of functionality.

### *E. FTPTs and Scenario Files*

Figure 6 shows the dynamic FTPT coverage for four different tests A-D, where the pairs A-B respectively C-D were generated with different scenario files. The *y* axis represents the FTPT (upper graphs) respectively the FTPT as part of all the events (lower graphs).

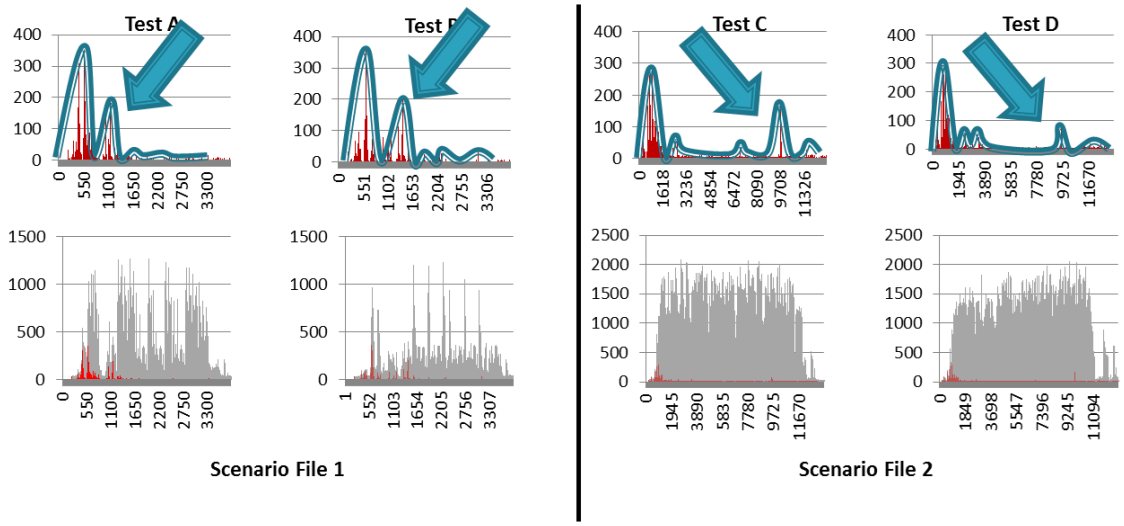


**Figure 6. FTPT and all events per cycle, for two different scenarios**

Tests generated using the same scenario file will have in common the contents and the functionality defined by the scenario. When running them, while they exercise the prescribed scenario, they generate similar FTPT patterns. The expectation of a certain similarity among tests generated with the same scenario file, drives our analysis to be made at the level of each scenario file separately, 12,000 such files for PowerPC core verification.

### *F. FTPT Waves and Their Windows of Opportunity*

For the same scenario we related a given wave with the targeted functionality. If there are hard to hit events in that wave, their likelihood of being hit is higher in a given cycle range, which is the window in which the wave can happen. For example in the case of a buffer full condition, there is a low likelihood of that condition to be fulfilled before a given number of cycles. This *window of opportunity* can be computed using experimental data. Figure 7 shows how twelve tests would point to a window of opportunity to hit an intended FTPT wave. With this the problem results in generating tests which have a higher likelihood of reaching the experimentally determined windows of opportunities.

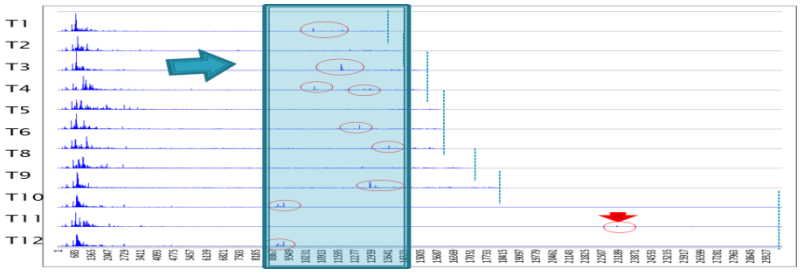


**Figure 7. Twelve tests and the window of opportunity for a given wave**

## VI. Data Analysis

### A. *Records for Data Analysis*

For each scenario we experimentally extract the dynamic FTPT coverage. For each scenario, for each event, the likelihood of an event to be hit before a given cycle is being measured from the gathered experimental data. The probability for an event to be seen in a test is the ratio between the number of simulated tests which contain event *e* $Nftpt(e)$, and the total number of simulated tests, $Ntests$ $P(e) = \frac{\text{Nftpt(e)}}{\text{Ntests}}$;

### B. *Probability Mass Function and Overall Probability*

***Cycles*** - the set of all cycles in which the event would have been hit and ***c*** is one such cycle.

Sample space ***S*** is the set of all possible outcomes, with ***s***, a possible outcome, $s \in S$.

The probability of event ***C*** to happen at cycle $c \in Cycles$, is given by a probability mass function $p: Cycles \rightarrow [0,1]$

$$p(c) = \Pr(C = c) = \Pr(\{s \in S: C(s) = c\}) ; \sum_{c \in \text{Cycles}} \text{p(c)} = 1;$$

An overall probability mass is being computed by applying Bayes' theorem and assuming independence between the probability to hit an event, and the cycle it hits the event.

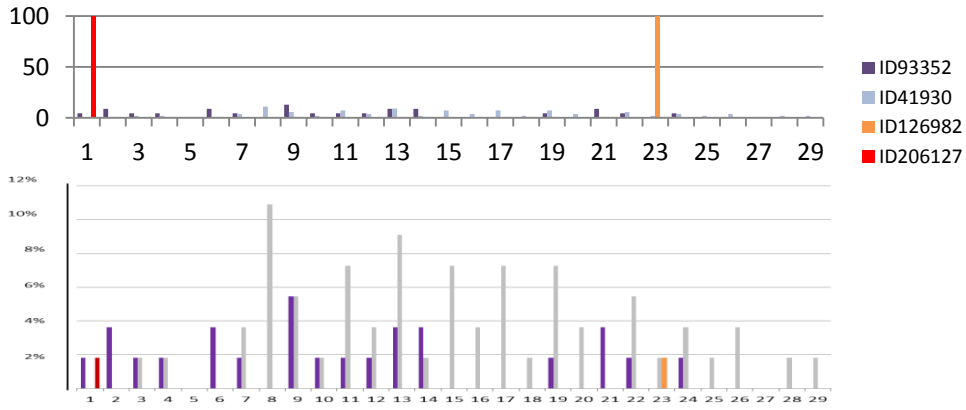


**Figure 8. Events probability mass function and overall probability mass**

Figure 8 is used to exemplify the probability mass function and the overall probability of four events for bins of 1000 cycles. The Overall Probability mass function shows the need to reach the cycle bin 23.

### C. *Overall Probability and Test Load*

For an event with a higher overall probability of being hit early in the simulation, the longer the test the more wasted resources, while for a signal (e.g. c) with the probability of being hit later in the test, the longer the test the higher the probability of hitting it. Using the experimental data we can compute a new number of cycles that would achieve an increased coverage efficiency as defined by the condition:

$x = C * \sum_0^x p(c)$ ;

Where $C$ – number of cycles ran, reflecting the original load, x is the new number of cycles and p( c ) is the probability mass function.

### *D. Hard-to-hit Events Only*

The above analysis is important for hard-to-hit events, defined as events hit less than 2k times for the moving window of the last 1M tests run. Out of 150k coverage events, approximately 73k are considered hard-to-hit, and 15k are never-hit.

## VII. Real Case Analysis

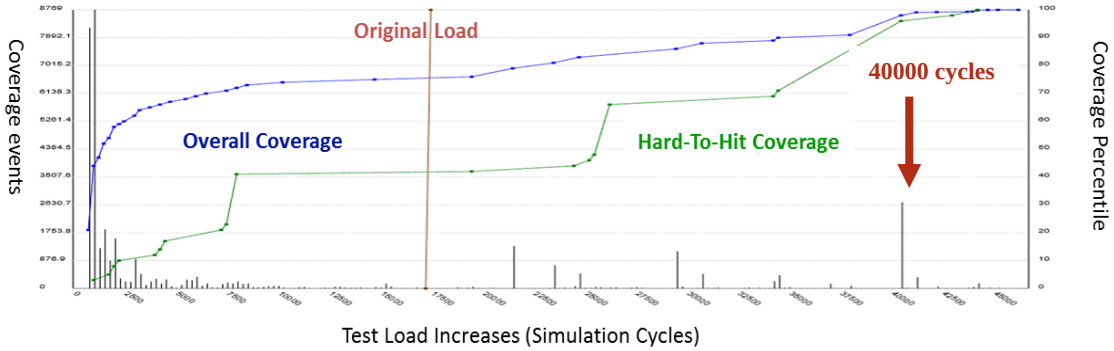


**Figure 9. Experimental coverage – test size dependency**

Figure 9 presents the experimental data as extracted from 40 identical test scenarios each with increased load. Each point in figure 9 is a test which ran as many cycles as shown by the x-axis, and for which the recorded coverage is represented by the y axis. The FTPT coverage is shown in blue, while green shows only the hard-to-hit events. The real life chart validates the assumption of waves. There are a few critical points (sizes) which allow for jumps in the coverage of the hard-to-hit events. For example there is a series of events which are available only around 40,000 cycles. Many coverage events, due to their inherent behavior, require a cumulative state to enable certain functionality.

The red vertical line shows the average size of the tests generated (directly related to simulation cycles run) for this particular scenario before the analysis. The analysis shows how that particular size was not able to trigger events which required a longer time to build, and which were seen only in longer tests. Increasing the number of instructions with which this scenario was generated succeeded in covering the events in fewer test simulations, and reaching never-hit events. Even though each such test was longer than before, they all were more likely to reach the wave of functionality intended by the scenario.

## VIII. Implementation Considerations

Recording the FTPT is expensive, and doing so for all scenarios takes large resources. The implemented adaptive system is based on a simple feedback loop for each scenario, which provides information regarding the change in the number of hard-to-hit events hit by a batch of tests generated with the same number of instructions (different for each scenario). The adaptive system keeps the number of instructions around a point of efficiency in achieving hard-to-hit coverage by controlling only one single variable of the test generation, the number of instructions in the test. Improvements can be easily made, by distributing the hard-to-hit events to the scenarios which are expected to generate them, and limiting the optimization to those events per scenario. For our specific implementation we simply determined the point of efficiency as the last point to exceed a hard-to-hit event threshold for the sampling

window. Even so, the implementation as is, achieved huge savings and changed the coverage efficiency for all scenarios.

## IX. RESULTS

For 12,000 test scenarios from IBM's POWER processor verification and for millions of tests generated since the simple adaptive system was implemented, the change achieved significant results:

- Decreased the number of events classified as hard-to-hit by 12%:73,000 to 64,000
- Achieved better coverage distribution, less redundancy on easy-to-hit coverage,
- Shifted manual work to the automatic process,
- Decreased never-hit before events by 13%: 15,000 to 13,000.

Overall it decreased the time to achieve targeted coverage and enabled finding bugs earlier, with savings of at least 18 person/months for the reduction in manual labor.

## X. CONCLUSION AND FUTURE WORK

Data analysis was used to observe the dynamics of coverage during testing. An analysis of model fitting and data distribution increased the understanding of how coverage is achieved in *waves*, which helps change the paradigm under which test generation is being managed. The test generation is to be focused on generating certain *waves*, and the likelihood of generating them is determined from experimental data using probability mass functions.

Following the lessons learned with the help of data summarization, a simple adaptive system was implemented and achieved immediate significant improvements to the overall efficiency of the verification process.

This is a first and immediate usage of the newly found understanding of how coverage advances in waves, and how we can increase the likelihood of reaching them. There is a large potential of using the distribution of signals within a wave to increase our understanding on signal correlations. This is also the starting point in a analysis per scenario and how different variations of the same scenario can increase the chances of hitting the desired wave, as a tool to guide the process of writing scenarios to target coverage (as in coverage driven test generation, but based on likelihood increase).

## REFERENCES


[1] J. Bergeron, A. Nightingale, E. Cerny and A. Hunter, "Coverage-Driven Verification," in *Verification Methodology for SystemVerilog*, Springer, Ed., 2006, pp. 259-280.

[2] A. Piziali, "CoverageDriven Verification," in *Functional Verification Coverage Measurement and Analysis*, Kluwer, 2004, pp. 109-136.

[3] M. Benjamin, D. Geist, A. Hartman and Y. Wolfsthal, "A study in coverage-driven test generation," in *Design Automation Conference*, New Orleans, 1999.

[4] S. Fine and A. Ziv, "Coverage directed test generation for functional verification using Bayesian networks," in *Design Automation Conference, 2003. Proceedings* , 2003.

[5] Y. Guo, W. Qu, T. Li and S. Li, "Coverage driven test generation framework for RTL functional verification," in *COmputer-Aided Design and Computer Graphics*, Bejing, 2007.

[6] Institute for Statistics and Mathematics of Wirtschaftsuniversität Wien, "The R Project for Statistical Computing," 31 10 2014. [Online]. Available: http://www.r-project.org/. [Accessed 08 11 2014].

[7] T. Benaglia, D. Chauveau, D. R. Hunter and D. S. Young, "mixtools: An R Package for Analyzing Mixture Models," *Journal of Statistical Software*, vol. 32, no. 6, 2009.

[8] mages, "Fitting distributions with R," 1 December 2011. [Online]. Available: http://www.r-bloggers.com/fitting-distributions-with-r/. [Accessed 8 11 2014].