Late Breaking Results: Test Selection For RTL Coverage By Unsupervised Learning From Fast Functional Simulation

Rongjian Liang*, Nathaniel Pinckney*
NVIDIA
Austin, US
{rliang, npinckney}@nvidia.com

Yuji Chai
Dept. Computer Science, Harvard University
Cambridge, US
yu927@eecs.harvard.edu

Haoyang Ren, Brucek Khailany
NVIDIA
Austin, US
{haoyangr, bkhalilany}@nvidia.com

Abstract—Functional coverage closure is an important but RTL simulation intensive aspect of constrained random verification. To reduce these computational demands, we propose test selection for functional coverage via machine learning (ML) based anomaly detection in the structural coverage space of fast functional simulators. We achieve promising results on two units from a state-of-the-art production GPU design. With our approach, an up to 85% RTL simulation runtime reduction can be achieved when compared to baseline constrained random test selection while achieving the same RTL functional coverage.

I. INTRODUCTION
Simulation-based methods are widely used for RTL verification. For a large digital unit, functional coverage closure can demand the simulation of millions of tests requiring months of runtime [1]. To reduce RTL simulation demand, in this work we select a subset of tests from a large test pool, typically created via constrained-random stimulus generation, so that the selected subset exercises most or all functional coverage previously hit by the entire test pool.

Previous work on simulation runtime reduction for coverage closure selected tests in the configuration or "knob" space of the stimulus generation environment [1] or directly in the generated input stimulus space [2]. However, many constrained-random tests may only be distinguished by the random seed fed into a stimulus generator, making anomaly detection on the input stimulus knob space ineffective. On the other hand, applying anomaly detection directly on the input stimulus space (DUT input bits × cycles) has only been tried for very small units or tests, as scaling these ML methods to large units can become intractable as the dimensionality of the input stimulus space increases. In this work, we explore a new approach to test selection guided by anomaly detection on the generated input stimulus. Our anomaly detection approach is complementary to other recently proposed ML-guided test generation methods [3], [4].

RTL is commonly developed alongside functional simulation models written in software languages such as C/C++. These functional simulators typically serve as a “golden reference” for constrained-random verification of RTL units, executing the correct behavior for checking RTL results on the same stimulus, but running 10 - 100× faster. In this work, we propose to use coverage data readily available from profiling these simulators to guide RTL functional coverage closure. Our contributions are summarized as follows:

- We find that functional simulator coverage can reveal meaningful information about RTL behavior with a tractable number of dimensions, enabling the use of anomaly detection guided test selection for RTL simulation.
- We apply an isolation forest anomaly detector for the first time to hardware verification and find it outperforms previous approaches.
- We demonstrate runtime benefits of our method for two units from a state-of-the-art industry GPU. For one unit with many infrequently-hit Cover Points (CPs), we find that 4K tests can hit 90% of the CPs hit by 20K tests, with 2× higher test efficiency than baseline random selection. For a second unit with fewer infrequently-hit CPs, an 85% RTL simulation runtime reduction is achieved with the same RTL functional coverage.

II. METHODOLOGY

A. Test Selection Flow
Figure 1 depicts our test selection flow. Fast functional simulations are run on a pool of tests, generating structural coverage reports, from which test representations are extracted. Next, isolation forest-based anomaly detection is applied in the representation space to select unusual tests that might exercise infrequent coverage events in the simulator. As will be shown in Section III, these anomalous events can correlate well with infrequently-hit RTL CPs. Only selected tests will go through RTL simulation, while unselected tests will be skipped for coverage closure runtime reduction.

B. Test Representation by Functional Simulation Coverage
The functional simulation coverage used as input to our anomaly detector can be automatically collected quickly and easily using software profiling tools as long as the functional models are written in standard languages such as C++. We collect coverage count data using the GNU GCC coverage mode along with GNU Gcov reports. Hit counts for every basic block (a straight-line sequence of code with only a single entry point and exit point) are extracted for each test. We empirically find that a large portion (over 50%) of basic blocks have constant hit counts across different tests. We prune such blocks since they do not provide any information for differentiating tests. After pruning, the hit counts for remaining basic blocks are stacked into a vector as the representation for a test.

C. Test Selection By Isolation Forest-based Anomaly Detection
Although supervised ML has shown promise in guiding stimulus generation for HW verification [3], these methods fail at finding difficult (infrequent or unhit) CPs in the training set. However, such CPs are critical to coverage closure. Given the strong correlation between our test representation and RTL coverage validated in
Section III-B, we argue that “anomalies” in our representation space tend to exercise unusual RTL functionalities, motivating our anomaly detection-based test selection method for finding difficult-to-hit CPs. We chose an isolation forest-based [5] anomaly detector due to its computational scalability in high-dimensional, data-rich scenarios and its robust performance. Figure 1(b) provides a visual depiction. An isolation forest is an ensemble of binary decision trees, called iTrees. Each internal node in a iTree represents a hyperplane that bi-partitions the representation space, while each leaf node represents one sample instance when an iTree is fully grown. As shown in Figure 1(b), anomalies tend to require less cuts to isolate them, leading to shallow leaf nodes in iTrees. Hence, the average depth of an instance in iTrees, after normalization, can serve as an anomaly score for the instance. The shallower the location of an instance in iTrees, the higher the anomaly score. Tests with top anomaly scores will be selected for RTL simulation.

III. Evaluation

A. Experimental Setup

Two units from an industry GPU in production were used to evaluate the proposed approach. The units were primarily datapath-oriented and contain tens of thousands of lines of code in the functional model and thousands of functional CPs (Table I). Around 20K tests were created for each unit using constrained random test generation. Synopsys VCS and Unified Report Generator, along with internal report metric scripts, were used to simulate RTL and gather functional coverage counts. From Figure 2 we find that many CPs of Unit A were hit by only a small percentage of tests, while Unit B had a smaller number of difficult-to-hit CPs.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Func. Lines</th>
<th>#CPs</th>
<th>#CPs hit by 20K tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>66248</td>
<td>17520</td>
<td>66248</td>
</tr>
<tr>
<td>B</td>
<td>31528</td>
<td>1266</td>
<td></td>
</tr>
</tbody>
</table>

B. RTL Coverage Prediction Results

Before deploying the anomaly detector, we first validated the correlation between the proposed test representation and RTL functional coverage. To do this, we train a 5-layer MLP model for each unit that inputs our test representation and predicts whether or not an RTL functional CP will be hit by the test. For each unit, 16K tests are used for training the MLP and inference is performed on the remaining 4K tests. After training, MLP models can achieve 95.9% inference accuracy and 0.987 area under receiver operating characteristic (ROC) curve for Unit A, and 88.9% accuracy and 0.960 area under ROC for Unit B, which indicates the effectiveness of our test representation. However, such a supervised learning method requires a large number of training samples and fails at CPs that are unhit in the training set, justifying the need for anomaly detection techniques that can handle difficult-to-hit CP rather than deploying the MLP directly to predict coverage.

C. Test Selection Results

As discussed in Section I, for the units in Table I, we are unable to compare to previous anomaly detection methods for test selection that use the input stimulus or configuration space directly as the test representation [1] [2]. However, we still implemented anomaly detection algorithms similar to these works with Pyod [6] and applied them to our functional simulator coverage test representation space. Specifically, the following methods are compared:

- IF AD (ours): isolation forest-based anomaly detection;
- OCSVM AD: One-class support vector machine-based detection;
- AE AD: auto-encoder-based anomaly detection;
- Random selection: averaged results of 100 random selection runs.

Figure 3 shows a comparison of test selection methods. In both units, anomaly detection dominates random selection. In Unit A, around 20% tests selected by our approach can cover 90% of the CPs previously covered by the 20K tests, while random selection needs 40% tests. In Unit B, only 1% of the selected tests are required to hit all the CPs covered by the 20K tests. When comparing anomaly detectors, isolation forest slightly outperforms the previous anomaly detectors (OCSVM and AE). It is also the most computationally efficient, handling 20K instances with up to 15K dimensions. Using Unit A as an example, the training & inference time for isolation forest, OCSVM and auto-encoder are 13, 167 and 23 minutes, respectively.

Figure 4 depicts the runtime analysis of our test selection approach. Our method is a lightweight technique, taking only 3% of the original test generation and RTL simulation time. In Unit A, with our approach, 25% of the original runtime is required to hit 90% CPs that are previously covered by 20K tests. In Unit B, 85% runtime is reduced while achieving the same RTL coverage.

### References


