Novel Test Detection to Improve Simulation Efficiency – A Commercial Experiment

Wen Chen, Nik Sumikawa, Li-C. Wang
University of California, Santa Barbara

Jayanta Bhadra, Xiushan Feng, Magdy S. Abadir
Freescale Semiconductor Inc.

***Abstract*—Novel test detection is an approach to improve simulation efficiency by selecting novel tests before their application [1]. Techniques have been proposed to apply the approach in the context of processor verification [2]. This work reports our experience in applying the approach to verifying a commercial processor. Our objectives are threefold: to implement the approach in a practical setting, to assess its effectiveness and to understand its challenges in practical application. The experiments are conducted based on a simulation environment for verifying a commercial dual-thread low-power processor core. By focusing on the complex fixed-point unit, the results show up to 96% saving in simulation time. The main limitation of the implementation is discussed based on the load-store unit with initial promising results to show how to overcome the limitation.**

I. Introduction

In the industry, full-chip functional verification remains largely relying on extensive simulation. In simulation-based verification, a state-of-the-art flow typically includes test generation, checking and coverage collection. The coverage metrics are defined to measure the completeness of the verification and direct simulation towards uncovered areas of the design [3]. A satisfactory coverage on a collection of metrics is required for tape-out. Various coverage metrics such as toggle coverage and functional coverage are supported in commercial simulation flows.

In a practical simulation-based verification environment, one of the most challenging tasks is to produce the tests that lead to the desired coverage level. One common practice is to manually produce direct tests targeting on specific coverage items. For processor verification, another common approach is constrained random test program generation in which users provide constraints and biases in the form of test templates and directives to the test generator [4]. The input to the test generator specifies the sampling scheme for various dimensions in the test space such as address selection, register dependencies, arithmetic data selection, etc.

This work is supported by Semiconductor Research Corporation, project 2012-TJ-2268.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA

Coverage-directed test generation (CDTG) is an emerging approach to overcome the test generation problem. CDTG techniques dynamically analyze coverage results and automatically adapt the test generation process to improve the coverage. Recent works proposed various techniques to learn from the simulation results and improve the test generation. These techniques employ a variety of learning techniques such as Bayesian Networks [5], Markov Models [6], Genetic Algorithms [7] and Inductive Logic Programming [8]. In [9] the authors propose an automatic target constraint generation technique to alleviate the burden of constraint generation. In [10] the authors proposed to learn test knowledge from micro-architectural behavior and embed the knowledge into test generator to produce more effective tests. In a latest work [11], the authors proposed a novel methodology to generate input stimulus and attain coverage closure for design verification based on GoldMine [12], a recently developed automatic assertion generation engine using data mining and formal verification techniques.

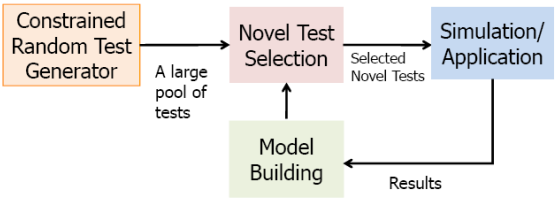

Fig. 1. Illustration of novel test detection

Novel test detection tries to tackle a problem much more restricted than CDTG. Figure 1 illustrates the approach. In a novel test detection framework, the assumption is that there is a constrained random test generator that can instantiate the test template to generate a large number of functional tests. The idea is to learn a novel test detection model based on the results from tests that have already been applied. This model is used to select novel tests from the large pool of tests before their application. Hence, only the selected novel tests are applied, which reduces the simulation cost.

The authors in [1], [13] proposed a novel test detection framework where Support Vector Machine (SVM) one-class algorithm [14] is used to build models. The framework is limited to analyzing fixed-cycle functional tests. The authors in [2] extended the application to build novel test detection models where tests are assembly programs and the context is for processor verification. The experiments were conducted based on a rather simple Plasma/MIPS processor design.

The objective of this work is not to suggest that novel test detection is better than any of the existing approaches or to compare them. In fact, novel test detection can be viewed as complementary to constrained random test generation and to CDTG. Our objective instead is to assess the applicability and effectiveness of novel test detection in a practical setting. We began by implementing the approach proposed in [2] in a company's in-house simulation environment for a dual-thread low-power processor. The targeted production date is in mid 2012 and the experiments were conducted in parallel to the ongoing verification efforts.

In this work, we explain the main findings based on the commercial experiment. These findings are organized into the remaining flow of the paper as follows:

- Section II presents simulation results to illustrate the existence of novel tests in the particular simulation-based processor verification environment.
- Section III reviews the approach proposed in [2], in particular the *graph-based kernel* method used to measure similarity on a pair of assembly programs. Applying the approach to the complex fixed-point unit demonstrates up to 80% potential saving in simulation time.
- Section IV discusses the major challenge of applying the graph-based kernel approach in practice. To implement the graph-based kernel demands a user to manually implement a *cost table* defining the similarity between every pair of instructions in consideration. To overcome this challenge, an alternative approach based on estimating coverage of an assembly program is proposed. This alternative approach implements a flow that requires minimal user involvement. The implementation is also easier. We demonstrate that this alternative approach can be as effective as the graph-based kernel approach and delivers up to 96% potential saving in simulation time.
- Section V presents how a novel test can be analyzed to extract rules to manually modify the test template. We present the benefit of this modification by showing coverage improvement on one experiment.
- Section VI discusses the main limitation of the alternative approach and proposes an extension to overcome the limitation. The effectiveness of this extension is shown based on an experiment on a module in the load-store unit with a potential 96% saving in simulation time.

  Section VII concludes the paper.

II. The Experimental Framework and Novel Tests

In this work, the experiments were conducted based on a dual-thread low-power 64-bit Power Architecture-based processor core. It is targeted to be manufactured in mid 2012 in a 28 nm technology. The processor core supports dual-thread capability that enables each core to act as two virtual cores. Each thread has dedicated Fetch, Decode, Issue, and Completion resources. Each thread also has a dedicated Branch Unit, Load Store Unit, and Simple Fixed Point units. The Complex Fixed Point unit as well as the Floating Point Unit and the vector engine are shared between threads. The core is designed with a memory subsystem supporting up to an eight-core implementation in a multiprocessing system.

The in-house simulation-based verification environment conforms to a state-of-the-art coverage-driven flow. An in-house test generator is used to generate constrained random test programs based on user-supplied test templates. During the test generation, architectural simulation is also performed and the simulation results are embedded in test programs. During the RTL simulation, the RTL simulation results are compared with the architectural simulation results for checking correctness. The coverage information is recorded and reported using a commercial coverage analysis tool. The verification coverage space is divided into subspaces. A subspace can be a part of the design, e.g., a particular unit or a specific mechanism such as memory collision, etc. In our experiments, we focused on the toggle coverage of the Complex Fixed Point unit (CFX) and the Load Store Unit (LSU). Test templates targeting on these units are provided by the verification team and are used in constrained random test generation.

A. Existence of novel tests in practice

Figure 2 shows three plots for three simulation runs, two on CFX and one on LSU. The x-axis shows the number of tests simulated, incremented by 30 at a time. The y-axis shows the normalized coverage based on the maximum coverage achieved for the respective unit in all experiments.

For the CFX, the first run consists of 2000 test programs each with 50 instructions and an initial machine state. The test programs are instantiated from a template based on 33 instructions targeting on the unit. The second run is similar, consisting of 10K test programs each also with 50 instructions and an initial state. For the LSU, the run consists of 3000 test programs each with 10 instructions and an initial state. The template is based on 6 instructions targeting on the unit.

In all three plots, we observe jumps in the coverage curves. These jumps are due to special tests that provide relatively significant coverage at the given simulation point. These special tests are the novel tests that we are looking for. If they can be identified before simulation, they can be applied earlier in the simulation run. As a result, the respective coverage can be achieved much faster.

Take the first plot as an example. We see that the jump occurs after simulating 1900 tests. We also see that the coverage curve is flat from 1300 to 1900. Suppose an engineer uses the template to instantiate 1600 tests and observes the flat curve. It is likely that the engineer would decide it is not effective to continue. Then, the coverage jump would have been missed. If we have the ability to predict novel tests before simulation, we can generate a much larger number of tests to begin with and consequently reduce the chance of missing a test capable of producing a significant coverage increase.

The three plots in Figure 2 shows the existence of novel tests in practical simulation-based verification scenarios. This gives a clear motivation to apply novel test detection to identify those tests before simulation.

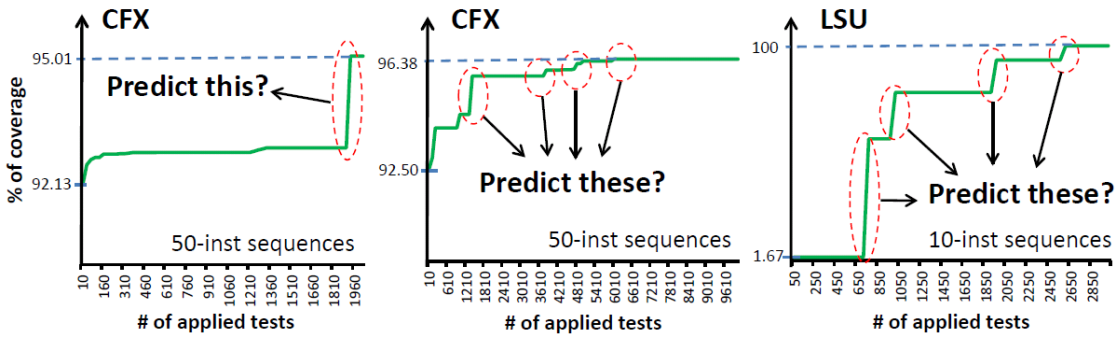

Fig. 2. Three simulation runs to illustrate the existence of novel tests

III. The Graph-Based Kernel Approach

A. Kernel based learning with SVM one-class

Support Vector Machine (SVM) one-class algorithm, such as the ν-SVM algorithm [14] is an unsupervised learning method that builds a model to identify outliers in a given set of samples. The parameter ν is a user-supplied input that represents an upper bound on the number of outliers and lower bound on the number of *support vectors*. In application with n samples, we typically set ν to be $\frac{1}{n}$ meaning that we want to build a model to incorporate at least $n-1$ samples, i.e. with at most one outlier.

In applying ν-SVM in novel test selection, the samples are tests that have been simulated up to the point of simulation. Suppose they are $t_1, ..., t_m$. A SVM model when applying to an un-simulated test T takes the following form:

$$M(T) = \sum_{i=1}^{m} \alpha_i K(T, t_i) - \rho$$

Conceptually, one can consider each α_i as a weight denoting the importance of test t_i in the calculation of the model. A test t_i is a *support vector* if $|\alpha_i| > 0$. Otherwise, it is a non-support vector, meaning that it is not used in the calculation. The ρ is a constant denoting the boundary of the measured outlier value for a test T. If $M(T) < 0$, T is deemed dissimilar to the simulated tests $t_1, ..., t_m$. The more negative the $M(T)$ is, the more dissimilar the test T is to $t_1, ..., t_m$. Given a set of un-simulated tests $T_1, \ldots, T_n$, let $M(T_j)$ be the most negative value computed by the model. Then, test T_j is the most novel test selected by the model.

The function $K(T, t_i)$ is called a *kernel* function used to measure similarity between T and t_i, i.e. a pair of tests. The novel tests selected by a SVM model highly depend on the definition of the kernel function. The kernel function dictates the perspective of what novelty means.

Suppose our objective is to cover a set S of coverage items. Suppose test T covers the subset S_T. Suppose test t_i covers the subset S_{t_i}. Intuitively, the similarity between T and t_i can be measured as $\frac{|S_T \cap S_{t_i}|}{|S_T \cup S_{t_i}|}$. For t_i, S_{t_i} is known. However for T, S_T is unknown because it has not yet been simulated.

The SVM one-class is a kernel-based learning method [15]. Such a method consists of two components, a kernel function used to measure similarity between a pair of samples and an optimization engine used to build the model. Figure 3 illustrates the learning approach.

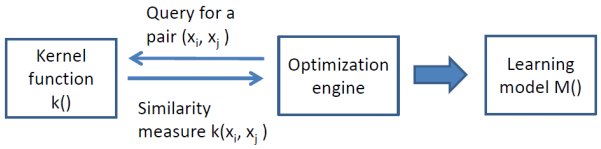

Fig. 3. Illustration of kernel-based learning

The SVM one-class algorithm concerns how to find the best values for $\alpha_1, \ldots, \alpha_m$ and ρ, based on a given kernel function. As shown in Figure 3, such an algorithm access the kernel function by querying the similarity between a pair of samples x_i, x_j. In application, one can alter the kernel definition without changing the SVM algorithm in order to influence the model building process.

B. The coverage-independent graph-based kernel

Developing an appropriate kernel is at the core of applying the kernel-based learning algorithm. In our application, tests are assembly programs. Hence, the kernel function $K()$ needs to measure similarity between a pair of assembly programs. The work in [2] proposes a graph-based kernel that computes a similarity measure by analyzing two assembly programs. It is important to note that such a graph-based kernel does not rely on any coverage information by a test in the calculation. Hence, it is a coverage-independent kernel.

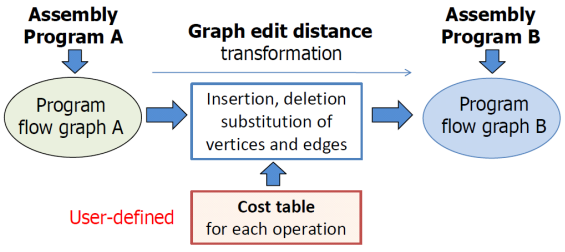

Fig. 4. The framework of computing graph-based kernel

Figure 4 illustrates the graph-based kernel. Each assembly program is first converted into a *program flow graph*, a directed graph capturing the possible execution flows of the program. Then, the kernel calculates the similarity between two programs based on the *graph edit distance* (GED) of the two graphs. The larger the distance is the more dissimilar the two programs are. The GED is measured as the minimal

cost of using a number of operations to transform one graph to the other. These operations include insertion, deletion, and substitution of vertices and edges. Each operation when performed has a cost value. The cost is defined in a *cost table*. For example, the cost of substitution of an addition instruction to a subtraction is smaller than the cost of substitution of an addition to a load/store instruction. This is because both addition and subtraction utilize the same execution unit while the load/store instruction utilizes the load-store unit.

Because the graph-based kernel is coverage independent, for a given cost table the process of building the model is fixed and consequently the novel tests detected by the model are fixed. This means that in order to apply the graph-based kernel to a given scenario, it is important to have a proper cost table. This cost table can be design dependent, unit dependent and coverage metric dependent. While this provides the flexibility to tackle a variety of scenarios, it can also be a challenge for its user to develop a proper cost table in practice.

C. Model building and novelty detection

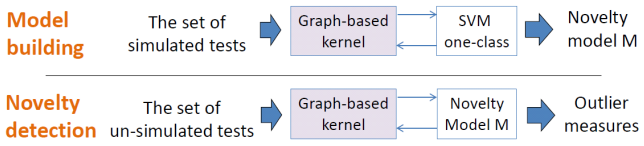

Fig. 5. The framework of graph-based kernel

Figure 5 illustrates the model building and novelty detection processes. In model building, a model is built on a set of simulated tests. In novelty detection, the model is applied to a set of un-simulated tests to calculate an outlier measure for each test. These measures are used to rank tests. The most outlying k tests are selected and simulated. For example, the process is iterative as shown in Figure 1 where in each iteration the most outlying k tests are selected for simulation.

D. Experiment results

The novel test detection framework using the graph-based kernel approach is implemented and integrated with the in-house simulation environment. Discussions in this section focus on the example shown in the first plot in Figure 2 before, i.e. the case with 2000 test programs for the CFX unit.

The novel test detection is applied iteratively where each iteration selects 30 tests to simulation from the pool of un-simulated tests. Figure 6 compares the coverage curves achieved with and without the novelty detection. The curve without is the same as that shown in Figure 2 before.

Without the novel test detection, the original simulation achieves a maximal coverage with 1930 tests. With the novel test detection, the same coverage is achieved using 190 tests, a 90% saving (i.e. 1-$\frac{190}{1930}$). The simulation time of 2000 tests is more than a day (using a single machine). This means that with the novel test detection, a day of single-machine simulation time can be reduced to less than two hours.

One may notice the huge coverage jump in the original simulation at around the 1930th test. This indicates a special

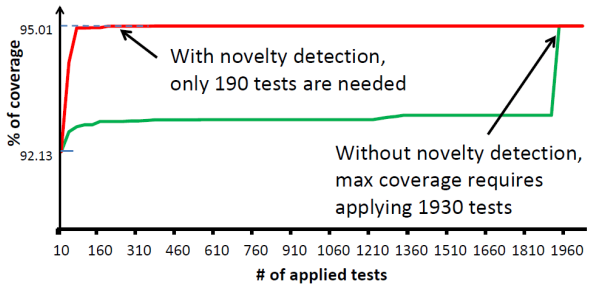

Fig. 6. Comparison of coverage curves with and without novelty detection

test whose characteristic is quite different from that of others, i.e. involving a dramatically different sequence of instructions. This might make the novel test detection problem easier. To assess the impact of this special test on the novel test detection, we conduct a different experiment by removing this test from consideration. In this revised experiment, we consider only the first 1800 tests.

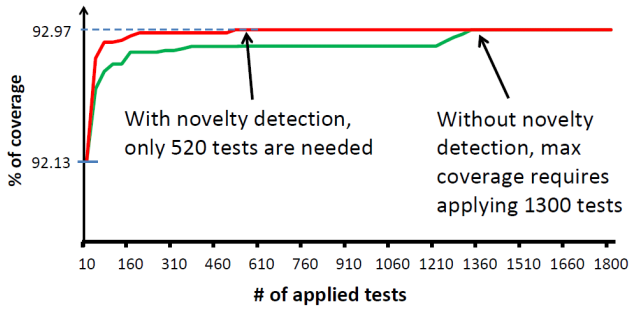

Fig. 7. Comparison of coverage curves with and without novelty detection based on only the first 1800 tests in Figure 6

Figure 7 shows the results with and without novel test detection based on the 1800 tests. Observe that in this case, the novel test detection can still provide a 60% saving (i.e. 1-$\frac{520}{1300}$). The figure also confirms that the existence of the special test does make the novel test detection more effective.

IV. Kernel Based on Estimated Coverage

A. Disadvantage with the graph-kernel approach

As discussed in Section III-B, the major disadvantage with the graph-based kernel approach is in the manual implementation of the cost table. Figure 6 and Figure 7 show promising results. However, these results were not obtained without noticeable effort to develop the cost table for verifying the unit. Such a development may take days or weeks to understand the behavior of each instruction with respect to the intended coverage space based on the target unit and/or design. Although one may argue that the development effort can be seen as a one-time cost, in practice, it represents a major obstacle for the acceptance of the approach.

B. Coverage-based kernel

To ease the use of the novel test detection approach, what we need is a new way to compute the similarity with minimal manual involvement. This motivated us to develop an alternative kernel method based on estimated coverage.

Recall from the discussion in Section III-A that a novelty detection model is of the form: $M(T)=\sum_{i=1}^{m} \alpha_i K(T,t_i)-\rho$ where $t_1,\ldots,t_m$ are simulated tests. Such a model is learned based on $t_1,\ldots,t_m$ to decide the values on $\alpha_1,\ldots,\alpha_m$ and ρ. To calculate the similarity between a pair of simulated tests t_i, t_j, i.e. denoted this kernel as $K_c(t_i,t_j)$, we can simply let $K_c(t_i,t_j) = \frac{|S_{t_i} \cap S_{t_j}|}{|S_{t_i} \cup S_{t_j}|}$, where S_{t_i} and S_{t_j} are subsets of covered items by t_i and t_j, respectively. Note that t_i, t_j are simulated tests and hence, S_{t_i} and S_{t_j} are known. Such a calculation can be based on a given set S of items to cover in the simulation. Hence, the kernel calculation only depends on the selection of S that is much easier to obtain than the cost table. For example, S can be the toggled lines in a specific module of interest. As another example, S can be a set of hard-to-cover toggled lines after some initial simulation.

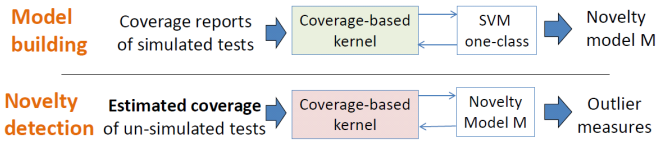

Fig. 8. The framework of coverage-based kernel

Figure 8 illustrates the framework using the coverage-based kernel. In model building, a coverage-based kernel works well because the true coverage of each simulated test is available. In novel test detection, the model M is applied to compute an outlying measure for each un-simulated test T. This requires computing $K_c(T,t_i)$ for each support vector test t_i where the true coverage of T is not yet known. Hence, to enable the approach, we require a method to estimate coverage for an un-simulated test T.

C. Estimating coverage before simulation

The idea to estimate the coverage of an un-simulated test is simple. Figure 9 illustrates the idea.

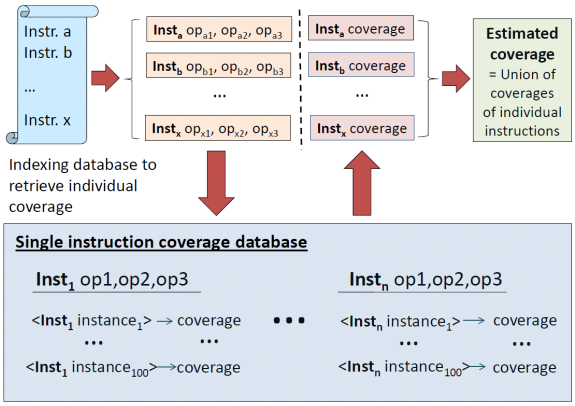

Fig. 9. Illustration of coverage estimation flow

For each single instruction, we randomly instantiated h instances using the constrained random test generation framework. In the experiments, we had $h = 100$. These 100 instances were simulated and their coverages were recorded in a database. There are 600+ instructions defined by the PowerPC ISA. It took about 250 hours to build the entire single instruction coverage database. The the storage requirement is about 480GB. The simulation time represents a one-time cost for the approach.

For a given un-simulated program T consisting of a sequence of instructions, for each instruction I we retrieve the coverage from the database based on the instruction instance that is closest to the instruction I. This closeness is decided based on an *indexing function*. We implemented the indexing function to look for the closest instruction instance based on Hamming-distance calculation between the operand values of the instruction I in T and the operand values of the instruction instances stored in the database. For each instruction I in T, the indexing function decides the closest instruction instance in the database. Then, the corresponding coverage is retrieved and used for I. To estimate the coverage of T, we simply take the union of all the retrieved coverages.

It is important to note that using the union operation to estimate the coverage presents a major limitation to the approach. This limitation will be discussed in Section VI later.

D. The accuracy of coverage estimation

To give an idea on the accuracy of the coverage estimation method, Figure 10 shows a result based on the 2000 test programs used in the experiment in Figure 6. The x-axis shows the accuracy measured in terms of the percentage of overlap between the estimated coverage and the true coverage of a test program. The average estimation accuracy is around 75% and is far from being perfect. Later in the experimental section IV-F, we will show that this accuracy is sufficient for novel test detection to be effective.

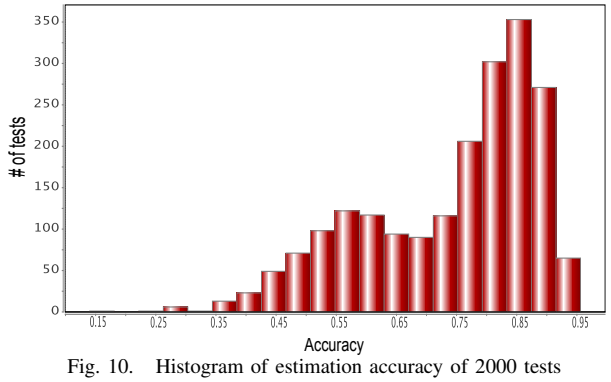

Fig. 10. Histogram of estimation accuracy of 2000 tests

E. Dynamically adjust the coverage base set S

In Section IV-B, we discuss the flexibility of the coverage-based kernel method. The coverage is estimated based on a set S of coverage items where this set can be flexibly defined. We call such a set the *coverage base set*.

Recall that novel test detection is an iterative process. Hence, ideally in each iteration the perspective of novelty should be defined with respect to the uncovered items. In other words, the novelty of a test should be evaluated based on its chance to provide coverage on the uncovered items.

Figure 11 illustrates the iterative process. Initially, a set of tests $T_1,...,T_n$ are simulated. Then a novel test detection model M_0 is learned from the coverage results of

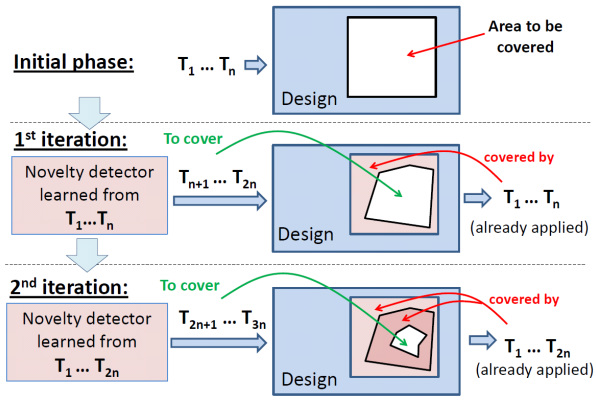

Fig. 11. An ideal iterative process with novel test detection

$T_1, ..., T_n$. When applying M_0 to select the next n novel tests $T_{n+1}, ..., T_{2n}$, we would like to cover the uncovered area in the design. To achieve this effect, we can perform the following adjustment on the coverage base set S.

Initially, suppose the set S contains p items $c_1, \ldots, c_p$. Let each item c_i be associated with a weight w_i initialized as 1. We calculate the coverage as $\sum w_i$ for all i such that c_i is covered by a test. Every time c_i is covered, w_i is adjusted to w_i/a where a is a constant such as $a = 2$. Such a weight adjustment scheme depreciates the importance of a covered item gradually.

Similarly, after the first iteration, a novel test detection model M_1 is learned based on all the simulated tests $T_1, ..., T_{2n}$. This model M_1 is used to select the next n novel tests $T_{2n+1}, ..., T_{3n}$ for hitting the uncovered area.

It is important to note in model building, those uncovered items do not participate in the coverage-based kernel calculation. This is because in model building, the true coverage of simulated tests is used and an uncovered item is skipped in the coverage calculation. When the model is applied to estimated coverage for an un-simulated test, an uncovered item may participate in the kernel calculation. This is because it is possible that the instruction instances retrieved from the database can hit the uncovered item.

F. Results compared to the graph-based kernel method

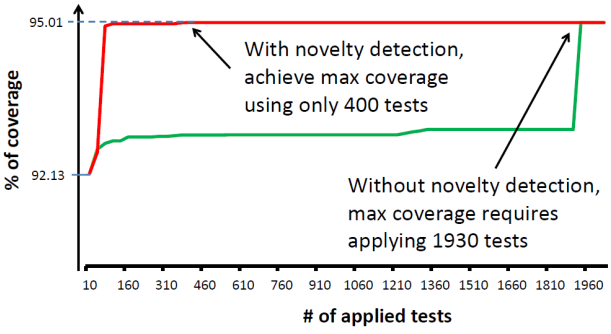

Fig. 12. Comparison of coverage curves with and without novelty detection using the coverage-based kernel; The same example shown in Figure 6 before

Figure 12 shows the result based on the same example shown in Figure 6 before. Again, each iteration the top 30 novel tests are selected for simulation. We see that with the novelty detection, only 400 tests are required to achieve the same coverage of using 1930 tests in the original simulation run, an 80% saving. Comparing this result to that shown in Figure 6, we observe the effectiveness is not as good as before. However, 80% remains a significant saving.

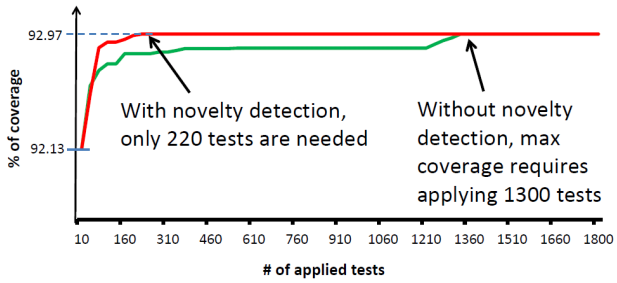

Fig. 13. Comparison of coverage curves with and without novelty detection using the coverage-based kernel; The same example shown in Figure 7 before

Figure 13 shows the result based on the same example shown in Figure 7 before, i.e. using only the first 1800 tests by removing the one special test giving the big coverage jump at the 1930th test in the original simulation run. We see that with the novelty detection, only 220 tests are required to achieve the same coverage of using 1300 tests in the original simulation run, an 83% saving. Comparing this result to the 60% saving shown in Figure 7, the effectiveness is better than before.

G. Result on simulation of 10K tests

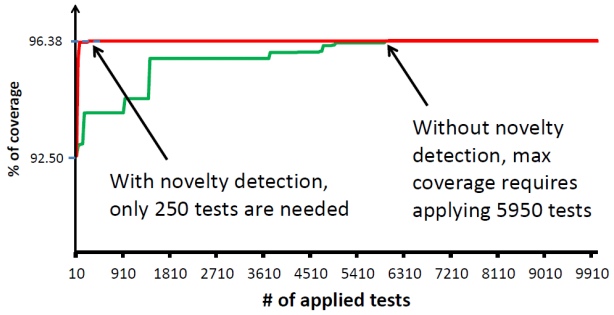

Fig. 14. Comparison of coverage curves with and without novelty detection based on the middle plot example shown in Figure 2 before

Figure 14 demonstrates the effectiveness of novel test selection using the coverage-based kernel for the 10k tests simulation example shown in Figure 2 before. Without novelty detection, the maximal coverage of the original simulation run is achieved with 5950 tests. With novelty detection, the same coverage is achieved using only 250 tests, or roughly a 96% saving. Simulation of the 5950 tests would have taken more than 4 days of single-machine simulation time. With the novelty detection, this time is reduced to less than 6 hours.

H. Two additional results

To show that the novelty detection approach can work well on tests based on a focused instruction base, we conducted an experiment using a test template based on only 6 CFX instructions. 2000 test programs were instantiated each with

50 instructions and an initial state. Figure 15 shows the results with and without novelty detection. Without the novelty detection, the original simulation achieves the maximal coverage with 1720 tests. With the novelty detection, the same coverage is achieved with only 100 tests, i.e. a 94% saving.

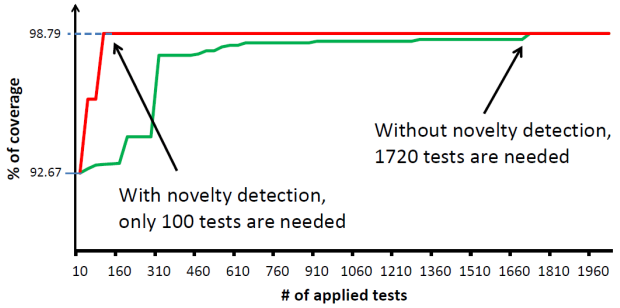

Fig. 15. Results based on 2000 tests instantiated from 6 CFX instructions

To show that the novelty detection can also work well on selected coverage points, we conducted an experiment by focusing on the 200 hard-to-cover points in the CFX unit. 2000 tests of 50 instructions were simulated in the original run. Without the novelty detection, the original simulation achieves the maximal coverage with 1930 tests. With the novelty detection, the same coverage is achieved with only 100 tests, i.e. a roughly 95% saving.

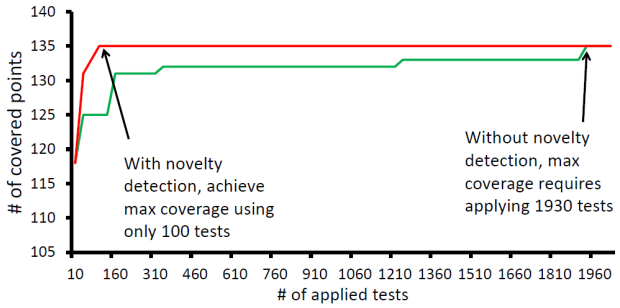

Fig. 16. Results based on 200 hard-to-cover points in CFX

V. Understanding a Special Test

Figure 12 earlier shows that the special test causing a coverage jump at about the 1930th test in the original simulation is captured by novel test detection within the first 100 tests. It is interesting to understand why the special test can cause such a coverage jump. To obtain this understanding, we employ a feature-based diagnosis approach.

A. Using a simple feature-based diagnosis scheme

In a feature-based diagnosis approach, a set of *features* are used to encode the characteristics of a sample (in our case a sample is a test). This encoding transforms each sample into a feature vector. Then, by analyzing the feature vector of a special sample against other non-special samples, we can extract rules to explain the unique property of the special sample, e.g. the special sample satisfies the rule and all other samples do not. For the rule extraction analysis, one can use a decision tree algorithm [16] or the subgroup discovery algorithm [17]. An example of feature-based diagnosis of design-silicon timing mismatch was presented in [18]. In our work, we apply the approach to analyze the special test to understand its specialty.

B. Manual test template modification

We implement a simple feature-based diagnosis scheme to understand the special test in Figure 12. We defined a set of features based on instruction types, operand values and the changes of those values in a program to describe the characteristics. We utilize the simple feature set to analyze the special test. The extracted rules are manually inspected and used to modify the test template trying to produce more tests similar to the special test.

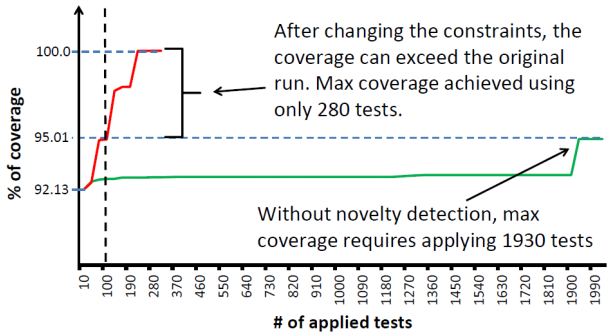

Fig. 17. Improving coverage by test template refinement

Figure 17 shows result of this test template modification. After simulation of the first 100 tests, the special test that results in a coverage jump is identified. After understanding the special test with the help of the simple feature-based diagnosis scheme, the test template is manually modified to produce additional tests. Observe that the additional 180 tests are able to improve the coverage to exceed that achieved by the original 2000 tests. The y-axis is normalized based on the maximal coverage achieved and that is why the best coverage shown is 100%. Note that this maximal coverage is also the best coverage achieved across all experiments shown in this work for the CFX unit and all coverages shown earlier are normalized based on this best coverage.

VI. Limitation of the Single-Instruction Database

Section IV-C discusses the method to estimate coverage for an un-simulated test program and points out its major limitation is in the use of the union operation to compute the coverage (also see Figure 9 for this union operation). Because the estimated coverage of a test program is the union of individual estimated coverages of all the instructions in the test program, such an estimated coverage does not consider coverage contributed by multiple instructions collectively. This limits the application of novelty detection to, for example, the load-store unit consisting of multiple finite-state machines, arrays and register files. For example, a data-forwarding event occurs when Read-After-Write hazards are present. Using the single-instruction database would be unable to properly estimate the coverage given by a test containing such hazards.

The LSU is one of the most complex units in the design. It is responsible for scheduling and managing the out-of-order memory operations. To illustrate the idea for overcoming the limitation we focus on an experiment based on the data-forwarding module used in the store queue. The result of the original simulation run is shown in the third plot in Figure 2. Below we discuss how to refine the novelty detection implementation to capture those novel tests shown in the plot.

The idea is simple. To overcome the limitation of using the single-instruction database, we build a database with a large number of test program instances each consisting of three instructions. Then, we use the coverage information stored in this 3-instruction database to estimate the coverage of test programs with a longer length. The indexing function in Figure 9 needs to be modified. In other words, the estimated coverage of a 10-instruction test becomes the union of coverages of several 3-instruction instances retrieved from the database.

Figure 18 shows the result of applying this extension to the particular example. Without novelty detection, the original simulation achieves the maximal coverage with 2590 tests. With novelty detection, the same coverage is achieved with only 100 tests, a 96% saving. Again, the coverage shown on y-axis is normalized based on the coverage achieved in the particular example and hence, it is shown as 100%.

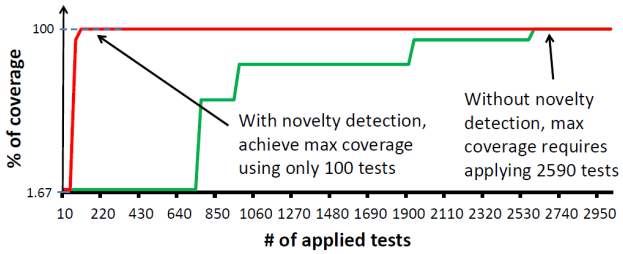

Fig. 18. Comparison of coverage curves with and without novelty detection using extended coverage-based kernel based on the third example plot shown in Figure 2

VII. Conclusion

In this work, we report the experience of applying novel test detection in a company in-house constrained random test generation and simulation environment for a Power architecture-compliant processor core. The first implementation is based on the graph-based kernel method. While this implementation can demonstrate 60-90% saving, its practical applicability is limited because of the requirement to manually construct the cost table. To overcome this limitation, a second implementation is proposed. This alternative approach is based on a coverage-based kernel method. The effectiveness of this approach is comparable to the graph-based kernel approach. The alternative approach demands minimal user involvement and hence is much more acceptable in practice. With the second implementation, we demonstrate 80-96% in various experiments. In one case, more than four days of single-machine simulation time can be reduced to less than six hours.

We discuss two extensions based on the second implementation. The first extension uses feature-based diagnosis to help a user to understand a special test identified early in a simulation run. Then, the test template can be manually modified accordingly to produce more tests similar to the special test. We show that this modification can lead to a higher coverage using much fewer tests than the original simulation run without novelty detection. The second extension overcomes the limitation of using the single-instruction database to estimate coverage. A new database of 3-instruction instances is added to capture coverage depending on multiple instructions collectively. The effectiveness of this extension is demonstrated on the data-forwarding module in the LSU with a potential 96% saving in simulation time.

References

[1] O. Guzey and et al., "Functional test selection based on unsupervised support vector analysis," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, june 2008, pp. 262 –267.

[2] P.-H. Chang and et al., "Online selection of effective functional test programs based on novelty detection," in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, nov. 2010, pp. 762 –769.

[3] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *Design Test of Computers, IEEE*, vol. 18, no. 4, pp. 36 –45, jul/aug 2001.

[4] J. Yuan, C. Pixley, A. Aziz, and K. Albin, "A framework for constrained functional verification," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, nov. 2003, pp. 142 – 145.

[5] S. Fine and A. Ziv, "Coverage directed test generation for functional verification using bayesian networks," in *Design Automation Conference, 2003. Proceedings*, june 2003, pp. 286 – 291.

[6] I. Wagner, V. Bertacco, and T. Austin, "Microprocessor verification via feedback-adjusted markov models," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 6, pp. 1126 –1138, june 2007.

[7] G. Squillero, "Microgp-an evolutionary assembly program generator," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 247–263, Sep. 2005.

[8] K. Eder, P. Flach, and H.-W. Hsueh, "Inductive logic programming," S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, Eds. Berlin, Heidelberg: Springer-Verlag, 2007, ch. Towards Automating Simulation-Based Design Verification Using ILP, pp. 154–168.

[9] H.-H. Yeh and C.-Y. Huang, "Automatic constraint generation for guided random simulation," in *Asia and South Pacific Design Automation Conference*, 2010, pp. 613–618.

[10] Y. Katz, M. Rimon, A. Ziv, and G. Shaked, "Learning microarchitectural behaviors to improve stimuli generation quality," in *Design Automation Conference (DAC), 48th ACM/EDAC/IEEE*, 2011, pp. 848 –853.

[11] L. Liu, D. Sheridan, W. Tuohy, and S. Vasudevan, "Towards coverage closure: Using goldmine assertions for generating design validation stimulus," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1 –6.

[12] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: Automatic assertion generation using data mining and static analysis," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 626 –629.

[13] O. Guzey and et al., "Increasing the efficiency of simulation-based functional verification through unsupervised support vector analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 1, pp. 138 –148, jan. 2010.

[14] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[15] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004.

[16] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[17] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski, "Subgroup discovery with cn2-sd," *J. Mach. Learn. Res.*, vol. 5, pp. 153–188, Dec. 2004.

[18] P. Bastani and et al., "Diagnosis of design-silicon timing mismatch with feature encoding and importance ranking - the methodology explained," in *IEEE International Test Conferencel*, oct. 2008, pp. 1 –10.