2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

ML based regression accelerator

Nikhil Singla
Google IT Services India Private Limited
nikhilsingla@google.com

Rohit Jindal
Google IT Services India Private Limited
rohitjindal@google.com

Pandithurai Sangaiyah
Google IT Services India Private Limited
pandithurai@google.com

Abstract - Complex designs require a comprehensive verification test suite, from basic sanity scenarios to complicated corner scenarios. As the verification cycle continues, tests keep getting stable with each RTL and DV milestone. Stable tests pass until or unless there is a change in that particular design feature and they end up using machine and tool resources without ROI. In this paper we are proposing an ML tool to pick "tests to run" algorithmically without compromising the regression quality. ML based regression accelerator is a self learning tool, which figures out the fragile tests which break frequently and prioritizes running those over stable tests thereby reducing the "tests to run" significantly. Algorithm ensures that each stable test is run periodically to catch any potential breaks and if the stable test fails, the algorithm automatically puts it in a relatively higher fragile test list. Also, the self-learning attribute of ML accelerator puts the fragile test into a relatively stable test list if it starts passing consistently over time in recent runs.

ML accelerator implements unsupervised K-means algorithm to cluster the testcases on the basis of which part of design the tests are hitting and along with the clusters it uses multiple data points like previous regression runs, coverage grading, delta changes in RTL, least run algorithm and mandatory runs, to decode the "tests to run" list.. This reduction results in efficient use of simulator licenses and compute resources which brings down the cost to chip. It also allows the user to dump any unexpected failure seen in the regression, without manually triggering the waveform dump. The solution proposed in the paper significantly left-shifts the verification cycle by reducing the turnaround time with full flexibility provided to users.

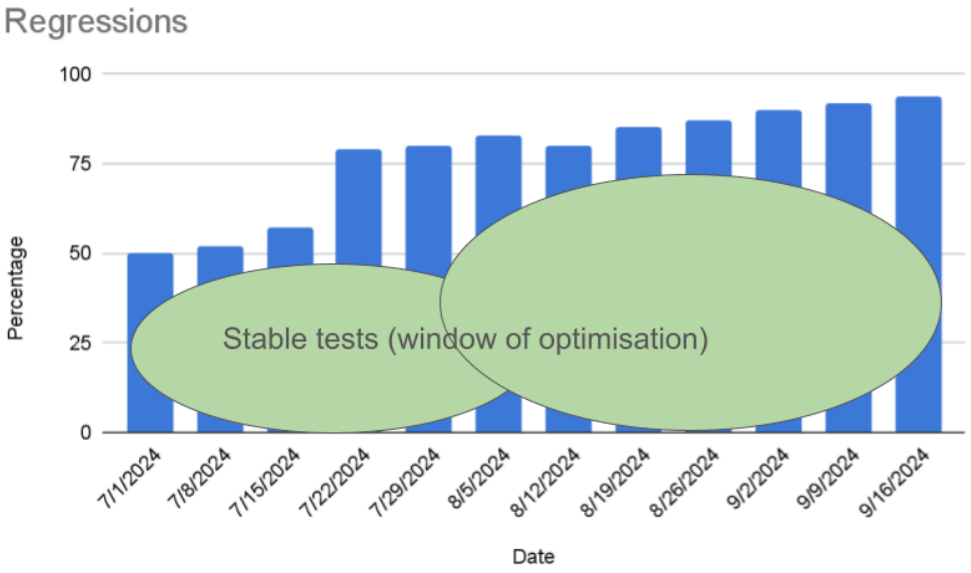
Introduction

ML accelerator is an algorithm to selectively narrow down the tests on the basis of multiple data points like historical runs where tests which are failing are given higher priority than always passing and coverage grading where tests contributing higher on coverage are kept higher in priority list. Any change in RTL is also tracked and only the tests impacted by the change are shortlisted. Accelerator is tuned to keep tests to run in check, if mandatory tests (failed, impacted by RTL change, coverage driven) are very high, algorithm reduces the passed test runs ratio to get efficiency and if mandatory ratio is low, more passed tests runs are picked to cover the full tests spectrum in fewer iterations. While filtering out the test cases algorithm maintains the test clusters to ensure tests finally shortlisted

are covering a major part of the DUT. ML K-means algorithm is integrated which bucketizes the tests on the basis of coverage numbers and test list decoding algorithm mandatorily pick tests from each bucket so that regression quality is not compromised during shortlisting.

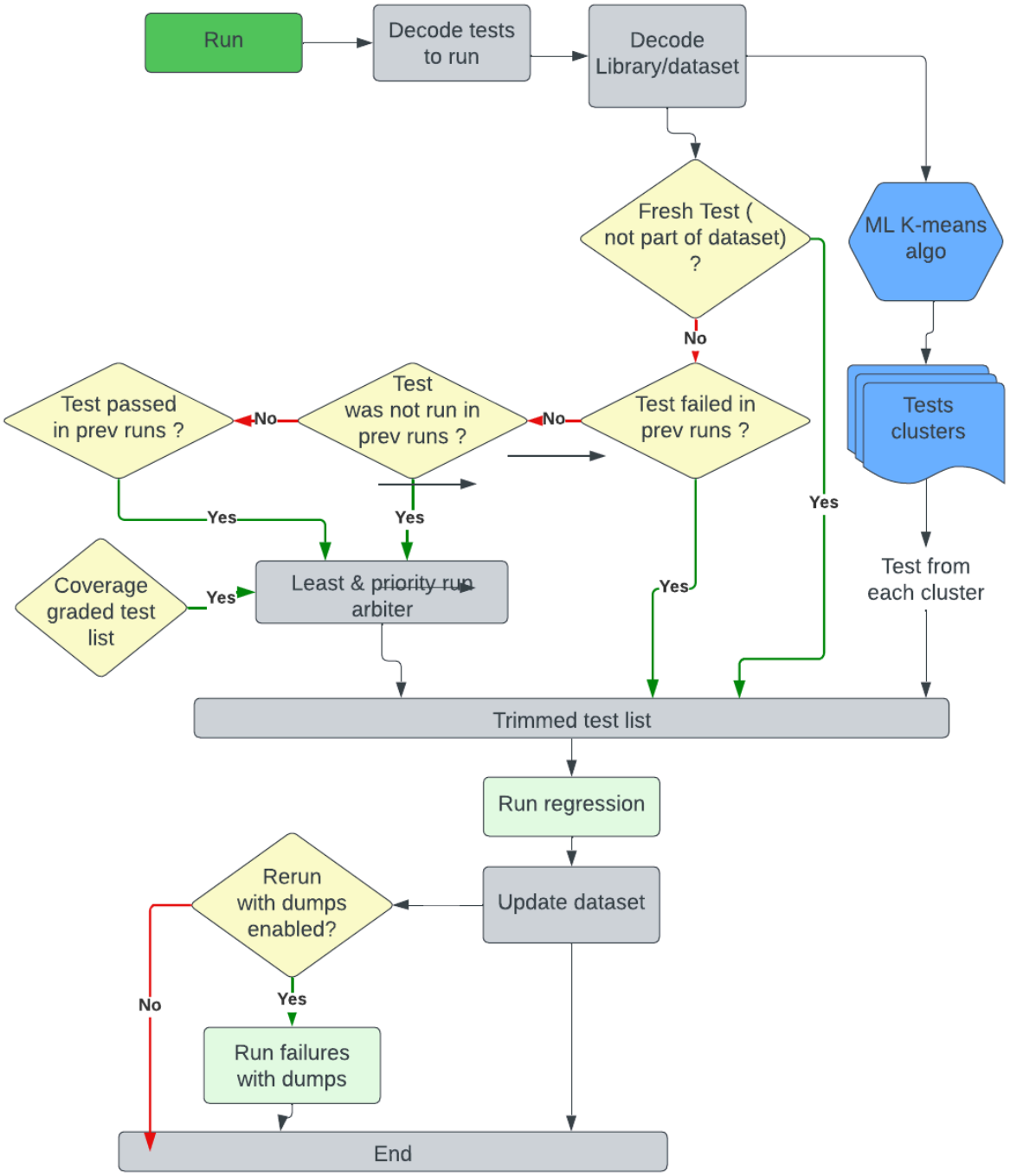
Features

- K-Means algorithm based clustering to bucketize the test cases on the basis of coverage
- Prune the tests on the basis of historical data
- Run subset of tests relevant to RTL changes
- Coverage grading awareness
- Mandatory test runs support
- Backward compatible, existing runs can be used for loading the dataset
- Dump the new failures for faster debugs
- Historical database can be refreshed if there is a major change in RTL
- Flexibility is given to the user to trim the tests to run aggressively
- Invalidate the last regression stats, it's useful in scenarios where regression results get corrupt due to infra issues



Implementation

Regression accelerator houses multiple algorithms and decision points. Whole algorithm can be viewed as a decision tree where depending on test categorization, the test is either picked or dropped from the next regression run. Also once the regression finishes, all the decision points are updated/tuned to make the algorithm more precise for the next run.



1

Clustering

While designing the test case filtering algorithm, one major criteria to consider is relative coverage. As a user, we don't want to pick the tests belonging to only one RTL feature and skip all others. Ideally in each regression, tests hitting as many nodes of RTL as possible should be picked.

To categorize/ cluster the test cases on the basis of similarity (in terms of which part of RTL code is targeted) .

ML based regression accelerator implements K Means clustering algorithm to bucketize the similar test cases which takes into consideration

---

1

- Coverage numbers for major RTL blocks
- Clustering takes care of cross across the blocks

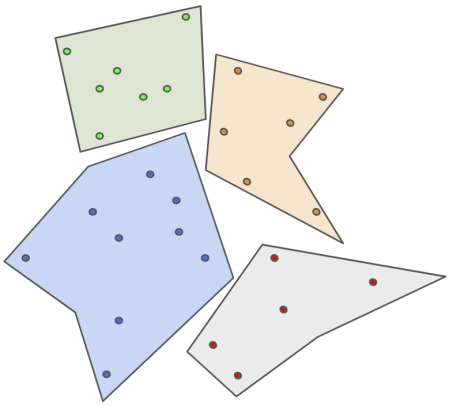
K-means clustering

K-means is an unsupervised learning algorithm, which bucketizes the items on the similarity. Steps involved in algorithm

- Major RTL blocks are identified in the design
- Functional cover groups clubbed on the basis of criticality and similarity
- Data structure per testcase is formed for coverage across each head (major blocks and functional coverage)
- Data structure is fed to K-means algorithm for clustering

	Code coverage			Functional		
Testnames	CPM	Cache	Core	CGs1	CGs2	Misc
Register	60	10	12	23	60	42
Boot	40	10	25	40	10	22
Power_txn	82	30	55	98	10	45
Virus	77	55	90	80	10	56
Performance	75	80	80	50	10	30
B2B rd/wr	75	45	60	50	10	30
Interrupts	40	10	25	40	98	20

K-means input trix



Example of test clusters

Algorithm data points

- **Historical regression runs**

  ML dataset (library) is maintained for all the historical runs, which serves as input to the accelerator to decode the tests to run for next regression. Failing tests and newly added tests are the highest priority followed by passing and skipped tests.

- **Coverage graded test list**

  Algorithms are modeled to arbitrate the tests on the basis of coverage graded test list as well. Users are required to dump the test list as per coverage contribution.When the graded test list is passed to the ML accelerator, it automatically prioritizes the tests which are hitting higher coverage.

- **Delta RTL changes**

  The algorithm caters to the testcase pruning when there is a change in RTL. To optimize the resources, it makes sense to run only the tests which are exercising the RTL code which changed. It makes use of a coverage database with tests info preserved and checks which test cases cover the delta change in RTL and picks them in the “tests to run” list.

- **Least run algorithm**

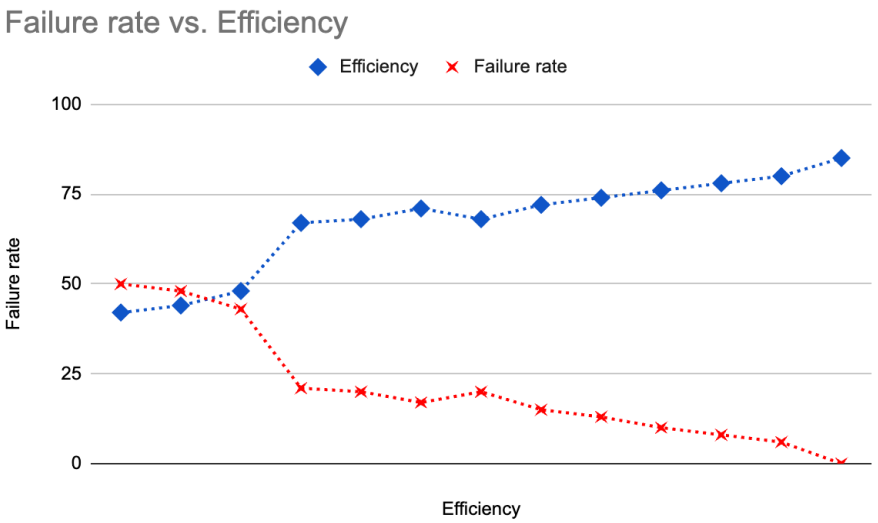
  The least run algorithm is used to pick the tests from a dataset which follow the same level of priority. Dataset currently uses 10 historical runs and from all the not run tests, the test which was least run will be picked for the next regression cycle.

- **User mandated tests**

  It is observed for many testbenches that there are a bunch of sanity test cases which the DV team wants to run every time and don't want that set of test cases to participate in any arbitration.

Results

ML regression accelerator is getting used for multiple testbenches, where efficiency as high as 75% is observed on 10% failure rate regressions. It has significantly reduced the simulator licenses and machine runtime. Savings go multifold if enabled at all the platforms. Generally license crunches are observed during the milestone closure due to multiple regressions running for all the platforms/blocks. Enabling the regression accelerator dropped down the traffic drastically resulting in minimal license queuing and faster regression closure.



Related work

Regressions are generally run automatically via jenkins or cron jobs, where full regressions are run and post processing is done to filter the failures of interest. There are many tools developed around triaging the failures and bucketizing it to make the data more readable for users but biasing the next regression on the basis of previous runs and data points is unexplored.
The concept put out in this paper is making runtime decisions, which saves the resources and CPU time.

Application

The ML based regression accelerator is built around the run script/ regression manager. The algorithm and the tool can easily be embedded into any DV setup with few changes in file formats. Currently, the setup has been enabled and verified across the platforms from IP/subsystem to SOC.
The infra changes required to incorporate the accelerator are minimal, as existing run directories can be used to train the dataset, no warm up runs are required.

The delta RTL change feature can be used for moving design from one chip to another. Algorithm will shortlist the tests impacting only the features which got changed, and negation of it can be leveraged to skip the tests for features removed.
The new failure dumping is highly recommended during the regressing phase near milestones, where the user doesn't want to dump all the tests due to dump size and runtime constraint. The selective dumping of new failures keeps the dumps ready for debugging without manual intervention.

Conclusion

The idea proposed in the paper is flexible and self-learning which applied to any verification flow will reduce the compute resources and simulation licenses significantly without impacting the quality of regression runs. The solution is already deployed in various testbenches and remarkable improvements in “tests to run” has been observed across the board. The solution is tapping the inherent nature of verification testbenches, where a mix of sanity and complex scenarios are coded throughout the DV cycle and test suite stability keeps increasing with every iteration/milestone.