2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

GraphCov: RTL Graph Based Test Biasing for Exploring Uncharted Coverage Landscape

Debarshi Chatterjee, Spandan Kachhadia, Chen Luo, Kumar Kushal, Siddhanth Dhodhi

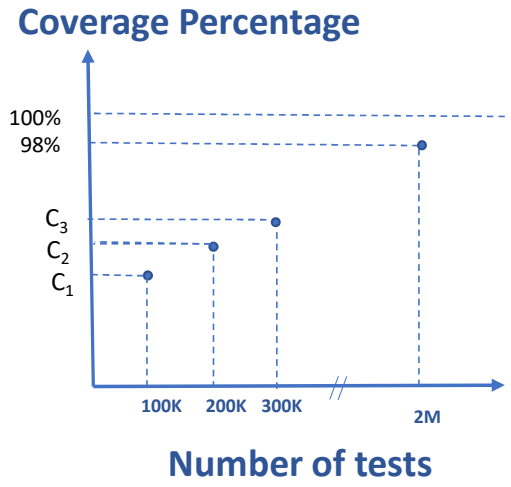
Nvidia Corporation

NVIDIA

accellera SYSTEMS INITIATIVE

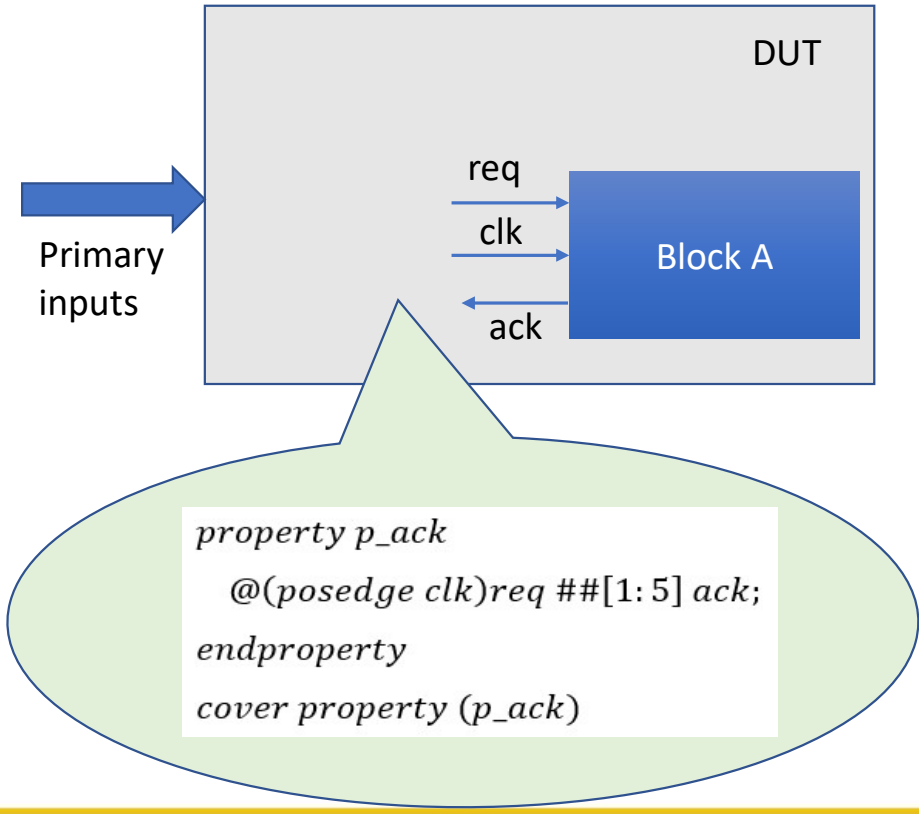
Introduction

- Coverage Closure is one of the most tedious phases of Design-Verification (DV)
- Towards end-of-project, coverage asymptomatically converges to a number less than 100%
- How do we cover the last mile?
  - Directed Tests
  - Fix Test-Bench (TB) Bugs
  - Waivers
- Can we expedite coverage closure time?



Background

- For this paper, Functional Coverage will be measured in percentage of Cover Properties (CP) that were hit
- CCP = Covered Cover Property
- UCP = Uncovered Cover Property
- Coverage% $= \frac{N_{CCP} * 100}{N_{CCP} + N_{UCP}}$
- Is there an automated way in which we can improve End-Of-Project coverage numbers by running a small set of tests?



Previous Work

- Considerable previous work exists on accelerated coverage closure
- Design2Vec: Uses RTL Control Data Flow Graph (CDFG) to generate GNN based model that predicts test coverage based on input knobs (NeurIPS, 2021)
- ML based techniques require collecting lot of data for building models
- Challenges:
  - Keep model updated with design changes
  - Learnings are not readily transferable from one unit to another
- This paper: Presents a simple heuristic for hitting Uncovered CPs without training models with huge amount of data

High Level Overview

- Identify Covered CP (CCPs) that meet three criterion:
  1. Overall hits to the CCP is relatively low compared to other CCPs in the design
  2. The CCP has large number of UCPs in its neighborhood
  3. The CCP is closely tied to the UCPs in its neighborhood
- Assign scores to identify CCPs that meet these criteria
- Call CCP with highest score as Target-CP
- High Level Idea: *If we can bias a set of tests towards the Target-CP, then we could possibly hit UCPs in its neighborhood*

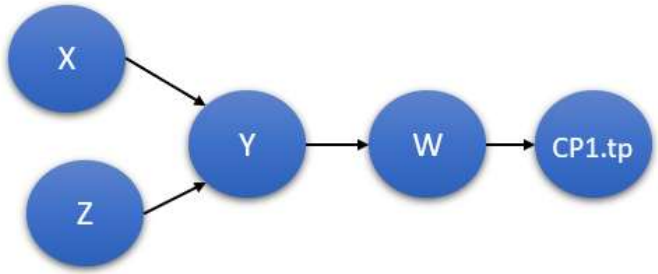
Trace Driver Graph

- Every node in TDG is either a CP or a Signal.
- A directed edge from source node (*S*) to a destination node (*D*) in TDG, implies one of the following:
  - 1) *S* is a driver for *D*
  - 2) *S* is a guard condition for the driver for *D*

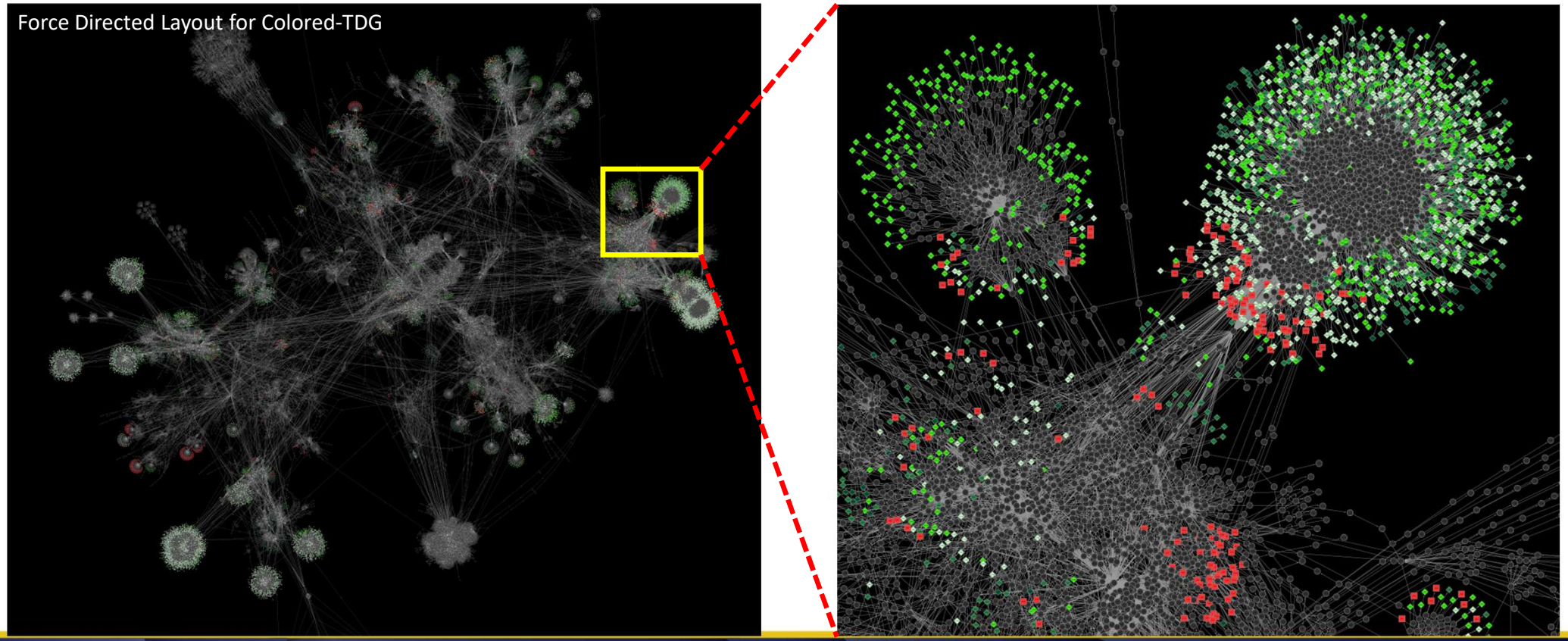
```
always_ff @(posedge(clk)begin
  if(X == 1)begin
    Y <= Z;
  end else begin
    Y <= 1;
  end
end

`ifndef SV_COVER_OFF
  wire W = (Y == 1);
  svt CP1(.clk(clk),.tp(W));
`endif
```

```
module svt(clk,tp);
  input clk,tp;
  wire clk,tp;
  property p1;
    @(posedge clk)tp;
  endproperty
  c1: cover property(p1);
endmodule
```



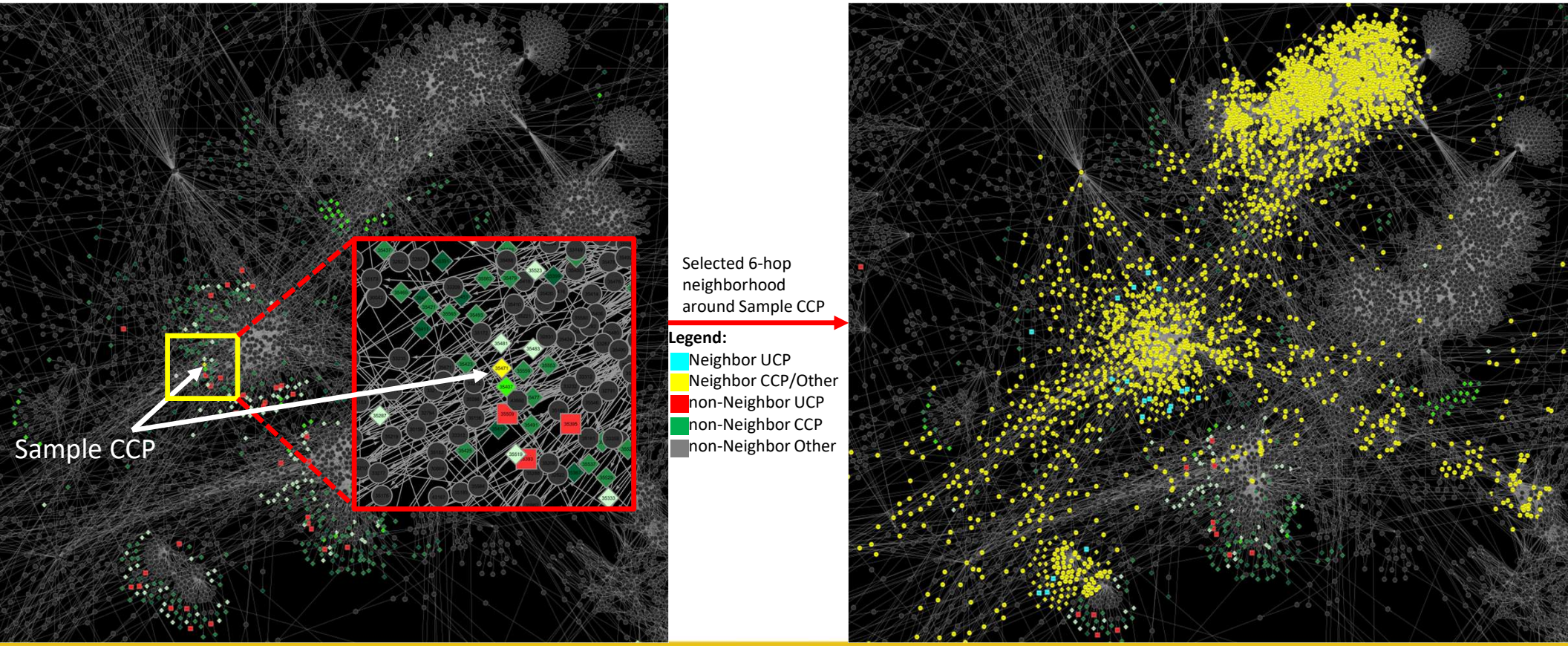
Colored-TDG from a Unit Level Design



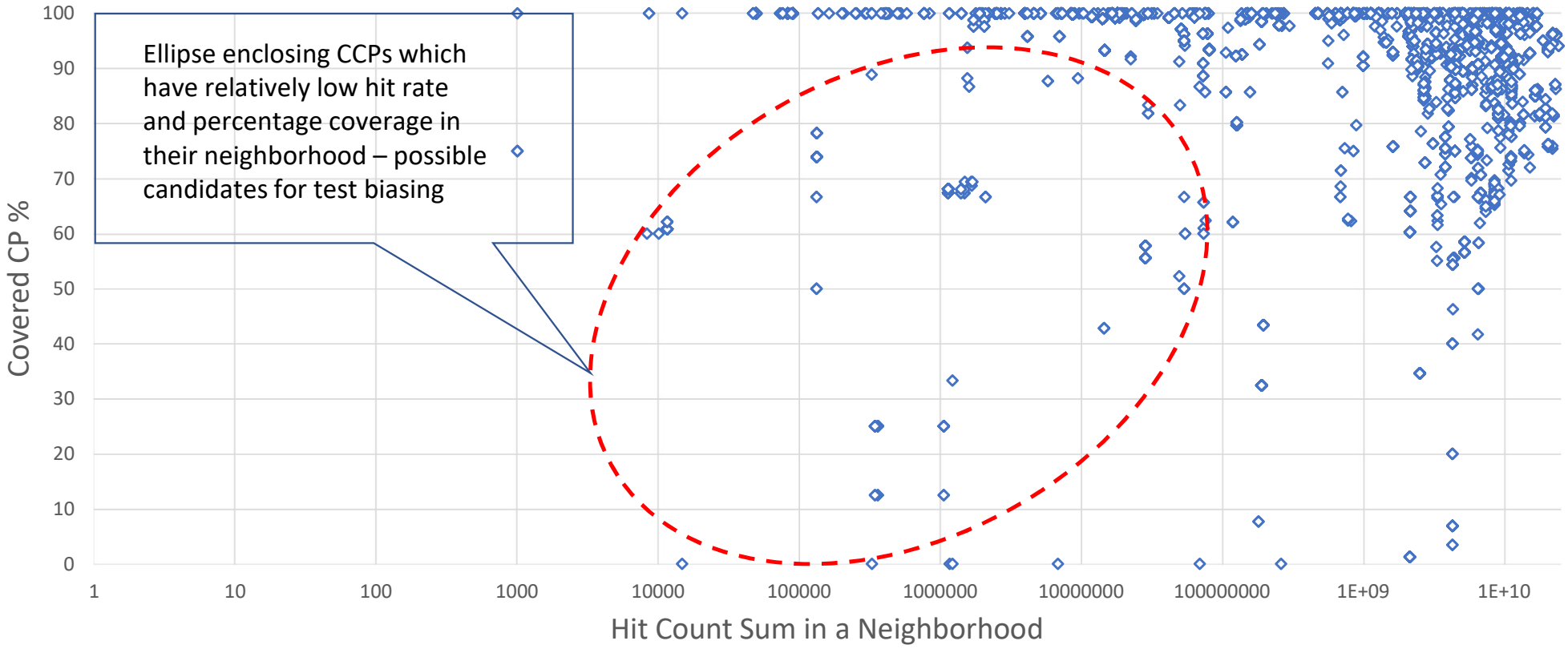
N-Hop Neighborhood

- N-hop neighborhood of a CCP:
  - *Set of nodes (Signals and CPs) that can be reached within N consecutive hops, starting from the CCP in the TDG*
- Alternative Definition:
  - *Starting from the CCP, if one is allowed to make N clicks to either Trace-Load or Trace-Driver button on Verdi, then all the signals and CPs that can be possibly reached, belongs to the N-hop neighborhood of the CCP*

6-Hop Neighborhood example of a CCP



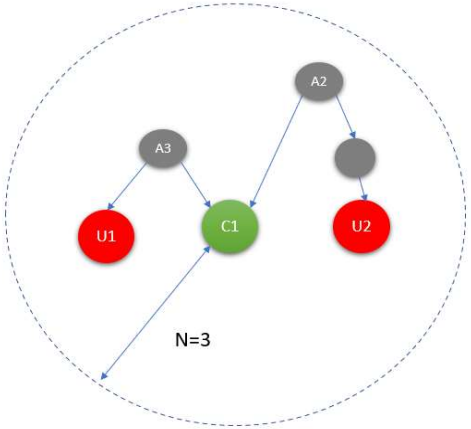
Some Stats

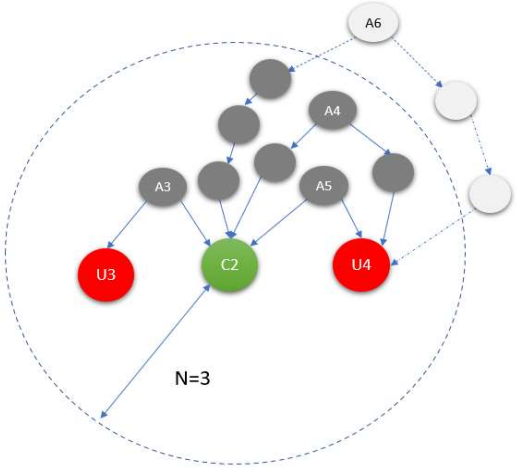

Ellipse enclosing CCPs which have relatively low hit rate and percentage coverage in their neighborhood – possible candidates for test biasing
Covered CP %
100
90
80
70
60
50
40
30
20
10
0
1
10
100
1000
10000
100000
1000000
10000000
100000000
1E+09
1E+10
Hit Count Sum in a Neighborhood

Assigning Scores to CCPs

- More shared ancestors means higher association
- Less distance of the UCP from the ancestors mean higher association
- Association between CCP and UCP = (A / d)
- Overall score is sum of associations

$$\text{Score}_{\text{CCP}_i} = \sum_{j=1}^{n} \frac{A(CCP_i, UCP_j)}{d(CCP_i, UCP_j)}$$



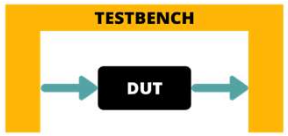


$$\text{Score\_C1} = \frac{1}{1} + \frac{1}{2} = 1.5$$

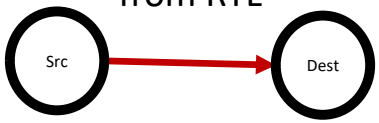
$$\text{Score\_C2} = \frac{1}{1} + \frac{2}{1.5} = 2.33$$

GraphCov Algorithm

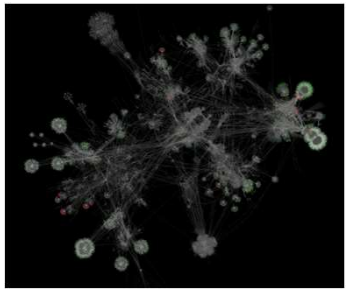
Step 1a:
Collect baseline coverage data
by running regressions



Step 1b:
Collect Signal Src-dest Map
from RTL



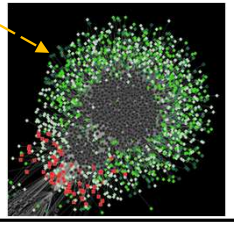
Step 2:
Generate Colored TDG



Step 3:
Assign scores to each CCP
(Filter out CCPs with high Hit count)

$$Score_{CCP_i} = \sum_{j=1}^{n} \frac{A(CCP_i, UCP_j)}{d(CCP_i, UCP_j)}$$

Legend:
- Uncovered CP
- Covered CP
- Other

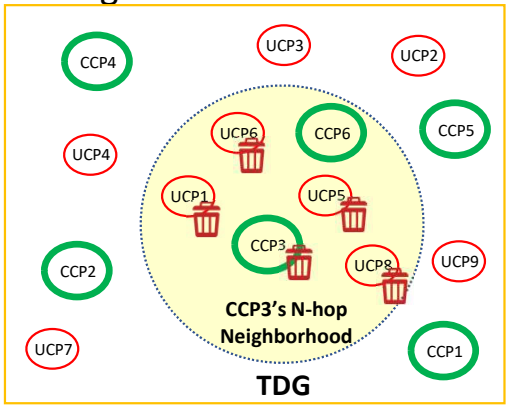


Loop to build **Target-CP list** with maximum UCP reach & association

Step 4:
Select CCP with highest score
and save it in Target-CP list

CCP3 > CCP5 > CCP2 > ...

Step 5:
Remove CCP with highest score and all UCPs in its N-hop neighborhood from TDG



Exit Criteria **not** met

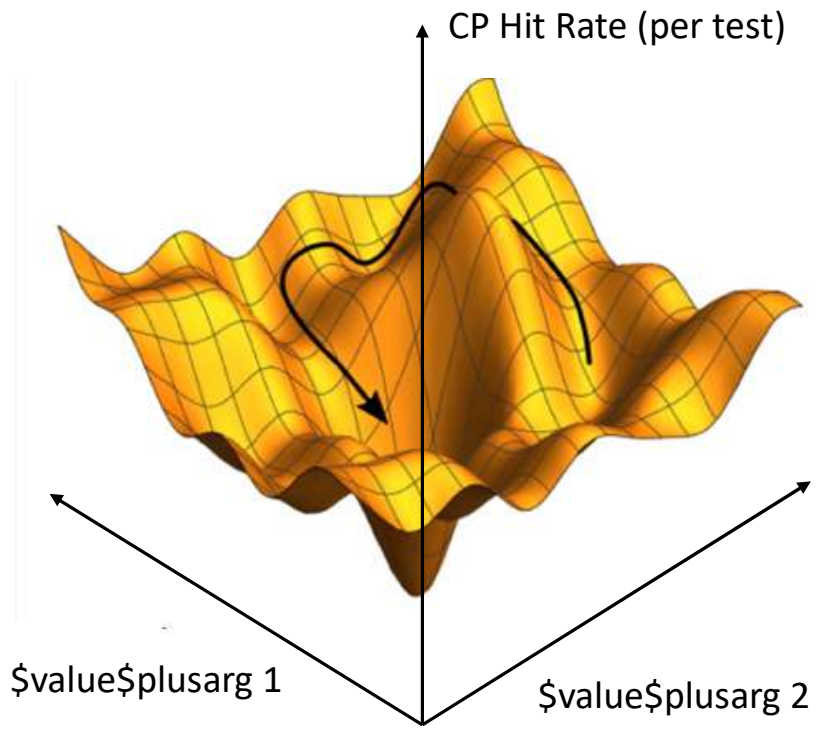
Step 6:
Exit Criteria
- No Low-Hit CCPs left
- No UCPs left
- Max Target-CP list size reached (user defined)
- Any other criteria

Exit Criteria met

Step 7:
Bias tests to hit Target-CPs list by using automated BayOpt tool or by manually setting test knobs

Biasing Towards Target-CP

- How do we bias tests towards a Target-CP?
  - Use in-house tool at Nvidia that uses Bayesian Optimization (BayOpt)
  - Alternative Approach: Ask DV engineers if there exists a set of $plusargs that can bias a set of tests towards a Target-CP
- BayOpt Limitations: Does not work well for large number of $plusargs
- No new $plusargs were added for the experiments



Runtimes

Steps	Runtime
Extracting Raw Data file for TDG	~1hr
Raw Driver Information to TDG	~3min
Run GraphCov to find top-M Target-CP	~5min
Total Time	~1hr8min

Results

Target-CPs	Tests Run	Boost CP Hit per test	Number of UCPs hit	Number of UCPs hit in 6-hop of Target-CP
Target-CP1	10000	23x	7	6
Target-CP2	5000	390x	24	22
Target-CP3	4000	20x	15	6

Summary: In 3 set of experiments using GraphCov, we were able to hit 46 UCPs and waive-off 44 UCPs. The number of CPs which were unhit and deemed reachable went down from 609 to 519, a 15% reduction.

Conclusion

- GraphCov shows promise in identifying UCPs which were unhit due to test biasing issues in the TB
- Future Work: Automate the process of knob selection. Current BayOpt approach cannot handle more than 20 $plusargs
- Limitations: Cannot hit UCPs which were unhit due to inherent limitations in TB code such as:
  - Over-constraints
  - In-built timing limitations in sequences etc.

Thank You!