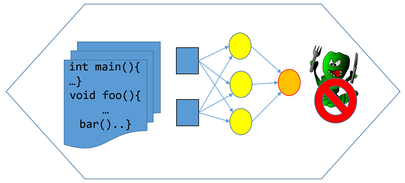
Cassandra: A Neural Network based Software Bug Detection Scheme

Mohammad M. Alam, Abdullah Muzahid
{mohammad.alam, abdullah.muzahid}@utsa.edu



Introduction

Programmers spent 50% of their effort in finding and fixing software bugs[1]. Software bug detection becomes more challenging due to prevalence of diverse platforms (e.g., multicores, many cores, data centers, accelerators etc.).

Cassandra uses neuromorphic hardware for invariant based software bug detection. Current schemes extract correct invariants of a program by analyzing many execution traces. The invariants are stored along with the program. At runtime, as each invariant occurs, it is checked against the stored invariants to determine if it is incorrect (and hence, buggy). Cassandra uses hardware based neural networks to perform online testing and training of invariants alternatively during a program execution. As a result, Cassandra can adapt to new inputs, execution environments, and even code with less execution overhead.

Motivation

```
S1: ifd = 0;                          /*Bug in gzip with an input "f1 - f2"*/
    while( .. )
      if( name=="-")
S2:     get_method(ifd)
        /*Process stdin*/
      else
S3:     ifd = open_input_file(..)
S4:     get_method(ifd);
        /*Process normal file*/
```

SOFTWARE BUG FACTS

Semantic Bugs & Memory related bugs:
- specific set of inputs
- specific sequence of executions and control flow

Heisenbug:
- depends on execution environment, compilation technique
- appears in one in millons of possible thread interleavings

Seq. of RAW

S1→S2	S3→S4
S3→S4	S3→S2
S3→S4	S3→S4
S3→S4	...
...	...
...	

- Data communication(RAW dependences) patterns are different in correct and buggy executions
- Neural Networks are efficient in learning patterns.
- Neural Networks are becoming popular as an alternative accelerator for different purposes.
- Can detect some unseen bugs after learning communication pattern.

How It Works?

Offline Training
- Profiling RAW(Read After Write) dependences from some correct executions
- Extracted RAW dependences are program invariants
- Train Neural Network with seq. of RAW as input in a sliding window

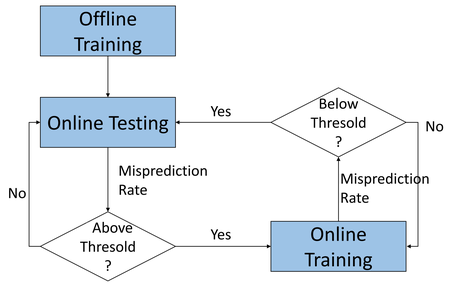


Fig: Feedback based Learning Design

Online Testing
- The neural network calculates its output for the RAW dependence sequence formed with last N RAW (S,L)
- (S,L) : each load(L) instruction of shared data and last store instruction(S) of the shared data forms (S,L).
- If the output is positive, the sequence is valid. Otherwise, the sequence is invalid.
- Invalid sequences (i.e., instruction addresses) along with the neural network output is recorded into ***Debug Buffer***

Online Training
- If misprediction rate is above certain threshold, Cassandra enters into Online Training mode

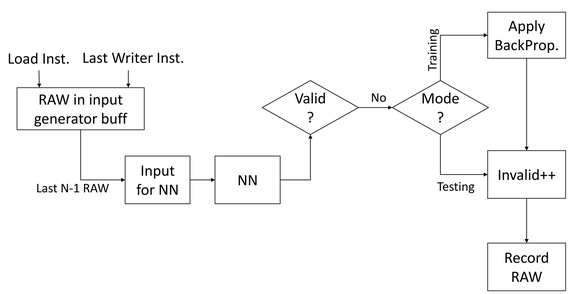


Fig: How one load instruction is processed

Neural Network in Hardware

- A fully configurable neural network time multiplexes an arbitrary network topology to a fixed number of neurons and incurs scheduling overhead
- We propose a **partially configurable** neural network with only one hidden layer. We limit the maximum number of inputs to a neuron to **M** (N_1 to N_m)
- Requires only one output(N_{out})from the neural network
- Use three stage **pipeline** without time multiplexing – S1(input), S2(hidden layer) and S3(output)
- Each of S2 and S3 takes T cycles where T is the number of cycles it takes for a neuron to calculate the output.
- If the FIFO is full, this pipeline takes an input(S1) after every T cycles. If the FIFO is not full, it can take an input in every cycle

Fig: Partially Configurable Low Overhead Neural Network

Offline Pruning and Ranking

- The Debug Buffer contains last few (e.g., 600) invalid RAW dependence sequences.
- Process the contents of the buffer in two steps - ***Pruning*** and ***Ranking***. The post processing is done offline after a failure.
- Pruning: Remove all the entries in the Debug Buffer that matches with **Correct Set**(inputs used for offline training or from input generator using any correct exections)
- Ranking: **Ranking algorithm** produces **higher rank** for a sequence if it has **more matched** dependences(comparing with Correct Set)
- More RAW dependences to match indicates that the mismatched dependence is closer to the root cause of the bug.

Evaluation

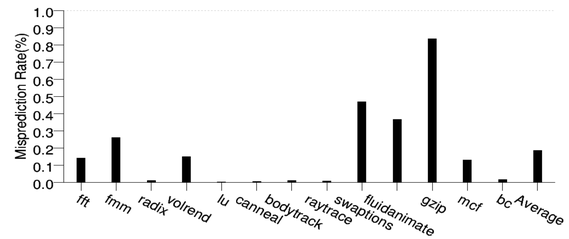


Fig: Misprediction rate with invalid RAW dependences

- Low false negative rate of 0.18%
- Accuracy of 93.84%(6.16% incorrect) for new code, whereas state of the art [2]is 0%(100% incorrect for code not seen)

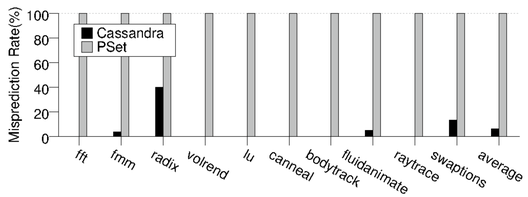


Fig: Misprediction rate with new code

- Overhead - 8.5%

Neural Network Parameters:
- Input FIFO– 8 entries
- 2 Multiplier-add unit for Neuron
- Sigmoid unit for neuron

Detects bugs in real-world applications

Program	No. of Bugs & Type
MySQL	3 atomicity violation bug
Apache	1 atomicity violation bug
Memcached	1 atomicity violation bug
Pbzip2	1 ordering violation bug
gzip	1 semantic bug
GNU coreutils	1 semantic bug & 2 buffer overflow bug

References

[1] Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer Katzenellenbogen. Reversible debugging software
[2] Jie Yu and Satish Narayanasamy. A Case for an Interleaving Constrained Shared Memory Multi-processor. In ISCA, June 2009.
[3] Brandon Lucia and Luis Ceze. Cooperative Empirical Failure Avoidance for Multithreaded Programs. In ASPLOS, 2013