Detecting Data Access Errors

Outline

- Introduction
- Data Access Errors
- Data Structure Signature
- Value Diversity
- Coding and Hashing
- Summary
- Future Work

Introduction

- Collect Data
  - Processor errors in space
- Error Detection
  - Software techniques
- Targeting Hardware Faults
  - Mostly transient errors
  - Can catch some software faults
- Software Implemented Fault Tolerance (SIFT)

Type of Errors

- Control Flow Errors
  - Wrong sequencing of instructions
- Data Transformation Errors
  - Wrong computation
  - e.g., ALU error
- Data Access Errors
  - Wrong value and/or address

Software Error Detection

- Control Flow Checking
  - Signature Analysis by Instructions (SAI)
  - Watchdog task
- Data Transformations
  - Stutter Step Mode execution
  - Assertions
  - Algorithm-Based Fault Tolerance (ABFT)
- Data Accesses

Data Access Errors

- Load/Store Errors
  - Not instruction fetches
- Error in:
  - Memory content (main or cache)
  - Memory address
- Bad Effective Address
  - Bit-flip in base register
  - Computation error
  - VA to PA translation error
  - Transients on bus
Detecting Data Access Errors

### Hardware Detection Techniques
- Data Value Errors
  - Parity and ECC
- Address Errors
  - Parity
  - Interleaving
  - HAL’s MMU [1]

### HAL’s MMU
- (72, 64) S4ED-SEC-DED
- Transient Address Errors
- Mix Parity of Address with Check Bits
  - $p$ = parity of address
  - $c[1] \oplus p, c[0] \oplus p$

### Software Detection Techniques
- Robust Data Structures [2-5]
- Data Structure Signature [6]
- Value Diversity
- Address-Hashed Data Values

### Robust Data Structures
- Redundancy in Data Structures
  - e.g., Doubly Linked Lists
  - Modified pointer implementation
  - Detect and correct single errors
- Used for Error Data in ARGOS
- Customized for Each Type
  - Not a general method

### Data Structure Signature (DSS)
- Assign a Unique Signature to Each Type
  - Type ID
  - Added before the first word of each instance
  - Always at address: base-1
  - Check for correct pointer
  - Added by compiler
- Software and Hardware Errors

### DSS Example
- With Existing Architectures
  ```
  Load Rx <- [p-1] /* get DSS */
  Load Ry <- ref_sign /* get reference signature */
  Comp Rx, Ry /* compare DSS, ref. sign. */
  Bneq error_handler /* signal error if not eq. */
  Load Rx <- [p+offset] /* OK, do usual access */
  ```
- With Architecture Support
  ```
  Load_DSS R0 <- [p-1] /* signature the DSS */
  Load Rx <- [p+offset] /* OK, do usual access */
  <or>
  Checked_load Rx <- [p+offset]
  ```
Detecting Data Access Errors

**Issues**
- All Instances Have Same Signature
- High Memory Overhead for Simple Arrays
  - e.g., array of integers
- Main Load/Store Instruction
  - Misses wrong effective address

**Observations**
- 32/64-bit Words
  - Unused bits (redundant)
- Common Values
  - 0, 1, -1, small integers
- Small Range of Integers
  - Loop counters
  - Status variables
  - Boolean: TRUE/FALSE
  - 'enum'
- Exploit Redundancy

**Encoding Data**
- Detect Wrong Value
  - ECC
  - High overhead in software
  - Simple encoding
  - Low overhead
  - Duplication
  - Different ranges
- Detect Wrong Address
  - Different encoding for each variable

**Value Diversity**
- Diversify
  - Counters: different ranges
  - Boolean: two random numbers
  - 'enum': multiples of a prime
- Check
  - Loop counter in range
  - Both values of a flag
  - Both 'if' and 'else' condition
- Detect Error in Control Variables
  - Wrong address loaded

**Address-Hashed Data Values**
- Save Address Info with Data
- Encoded Data
  - Range check
  - Duplication
- Hashed Values Stored in Memory
  - Hash with VA before store
  - Unhash value after load
  - Check the encoding
- Transparent to the program

**Hashing Example (1)**
- Direct Addressing
  /* original code */ /* modified code */
  Store [p] <= Rx Xor Rx <= Rx, p /* hash */
  . Store [p] <= Rx
  . .
  Load Rx <= [p] Load Rx <= [p]
  <check Rx> Xor Rx <= Rx, p /* unhash */
  <check Rx>

- Hardware Support
  Hstore [p] <= Rx Hload Rx <= [p]
Detecting Data Access Errors

Hashing Example (2)

Indexed Addressing

```
/* original code */

Store [p+offset] <- Rx
Add q <- p, offset
Xor Rx <- Rx, q
.
.
Load Rx <- [p+offset]
Add q <- p, offset
<check Rx>
```

/* modified code */

```
Store [q] <- Rx
.
.
Load Rx <- [q]
Xor Rx <- Rx, q
<check Rx>
```

- Or just use base address
- Same problem as DSS but checks actual load

Issues

- Initialized Data
  - Set at compile time
  - Modify loader, or
  - Initialization code
- Block Move
- I/O, DMA
- Multiprocessor Systems
  - Different VA for shared data
  - Use lower bits

Summary

- Maximize Error Detection Coverage
- Types of Errors
  - Each technique targets a subset
- Programming Practices (VD)
- Compiler-Assisted SIFT (SAI, DSS, VD?, AHDV)

Future Work

- Implement Compiler-Assisted SIFT
  - Modify C compiler
- Measure
  - Performance overhead
  - Memory overhead
- Fault Injection
  - Coverage

References (1)


References (2)