

# Unified Architectural Support for Soft-Error Protection or Software Bug Detection

Martin Dimitrov and Huiyang Zhou



School of Electrical Engineering and Computer Science  
University of Central Florida



# Motivation

- It is a great challenge to build reliable computer systems with buggy software and unreliable hardware
  - Software bugs account for **40%** of system failures
  - With technology scaling, transient faults are predicted to grow **at least in proportion** to the number of devices



# Proposed Approach

- **Observation:**
  - Soft-errors and software bugs manifest in similar ways.
  - program localities can be used to detect abnormal behavior (soft-errors or bugs) during program execution.
- **Unified Architectural Support:**
  - Utilize a type of value locality – Limited Variance in Data Values (LVDV) to detect abnormal behavior
  - A cache-like structure tracks LVDV locality and detects violations



# Presentation Outline

- Using localities to detect abnormal behavior
- Limited variance in data values (LDVD) locality
- Proposed architectural design
- Locality based soft-error protection
- Locality based software-bug detection
- Summary



# Using Locality to Detect Abnormal Behavior

- Execution results of instruction A:

20, 20, 20, 20, ... , 33554452



- Execution results of instruction B:

1, 2, 3 1, 2, 3 ... 1, 2, 3, 4



- Violation of the constant or stride locality hints the possibility of an error, or a software bug.



## Limited Variance in Data Values (LVDV)

- Variance between two values is defined as a simple XOR
- The variance specifies the **fraction of bits which vary** between the two values – the rest of the bits do not change and can be protected









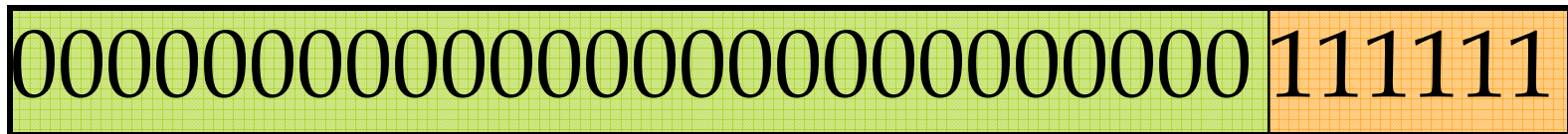


## An Example of LVDV Locality

- Execution results of instruction A:

1, 60, 12, 40, 2, ...

- No apparent pattern! However, the values are usually within a certain small range.



portion of result bits protected by LVDV





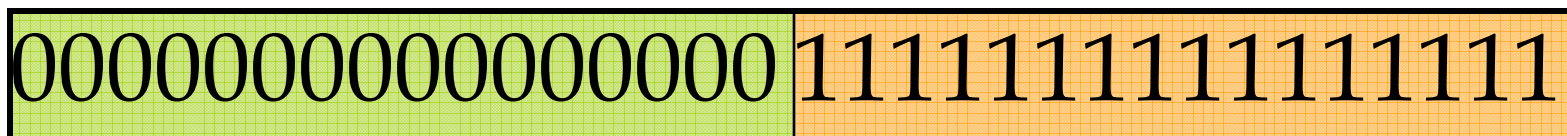
## LVDV Captures Region Locality

- Memory addresses produced by instruction A:

0x11112654, 0x11117838 ...

0x11111200, 0x11119088, 0x01119088

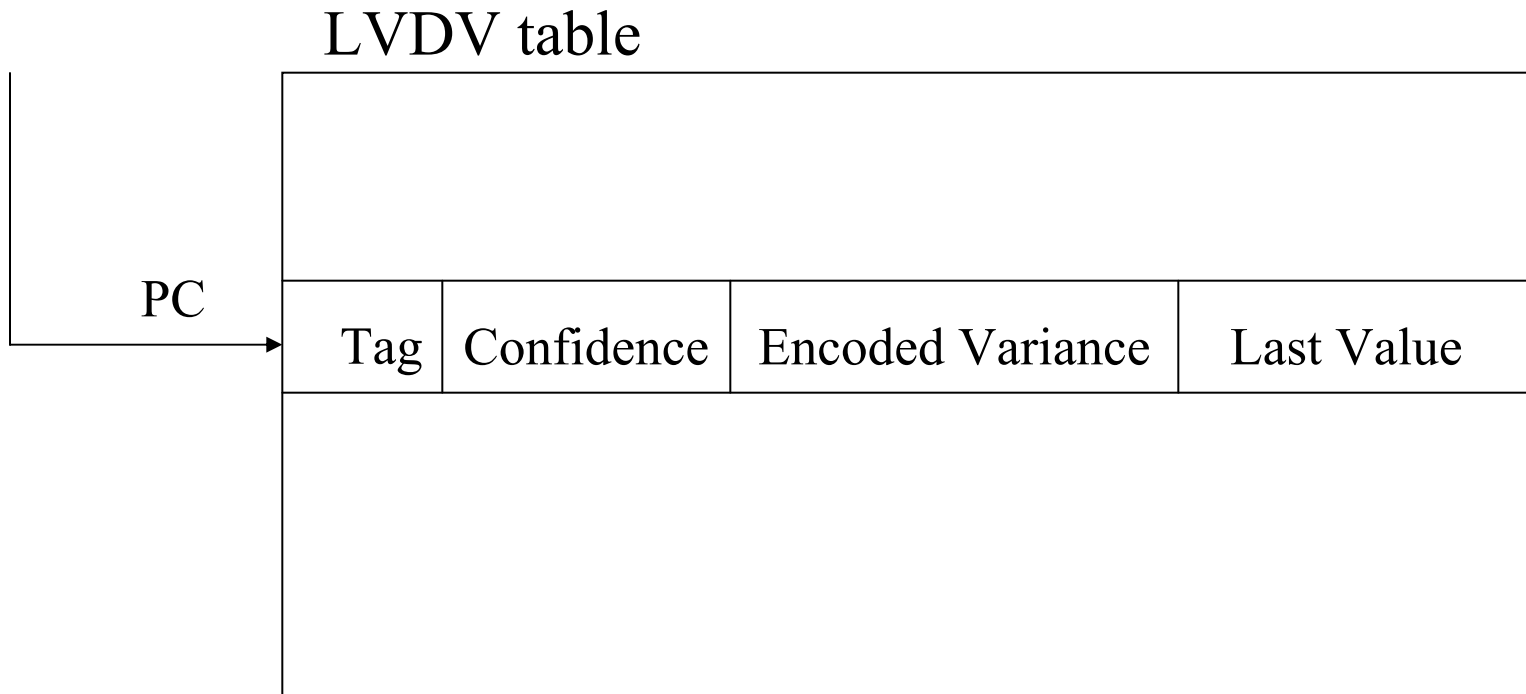
- Heap memory addresses produced by instruction A exhibit no stride locality.



portion of result bits protected by LVDV



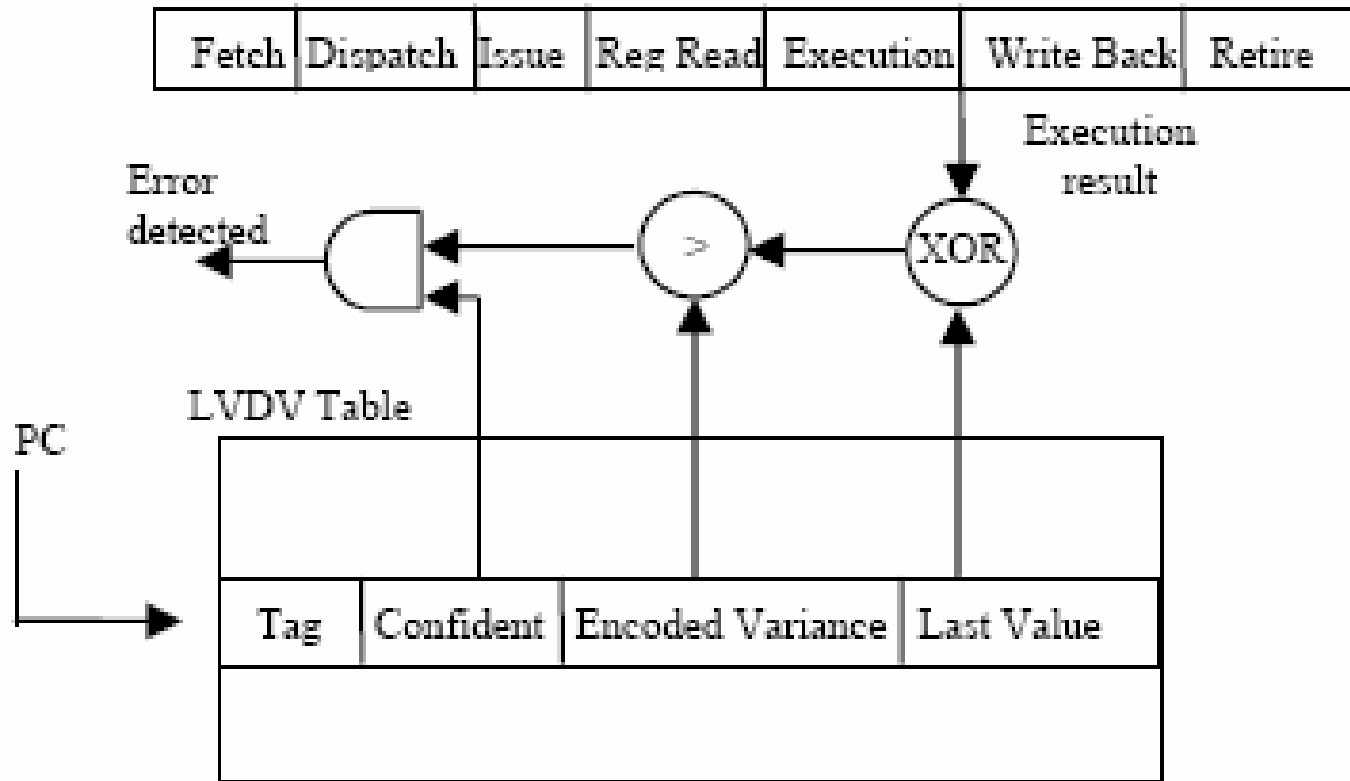
# Proposed Architectural Support



- The main structure in our architecture is an LVDV table.



# Interaction of the LVDV table with the processor pipeline





# Locality-Based Soft-Error Detection

- When an anomaly is detected, squash the pipeline as in a branch misprediction
- The anomaly is detected **promptly** and re-execution fixes the soft-error
- A single LVDV table can catch soft-errors in the computational logic, control logic – such as decoder, renaming table, issue queue and operand selection logic



# Benefits of Locality-Based Soft-Error Detection

- Provides information redundancy (no redundant execution required)
- Provides efficient, opportunistic protection of multiple hardware structures

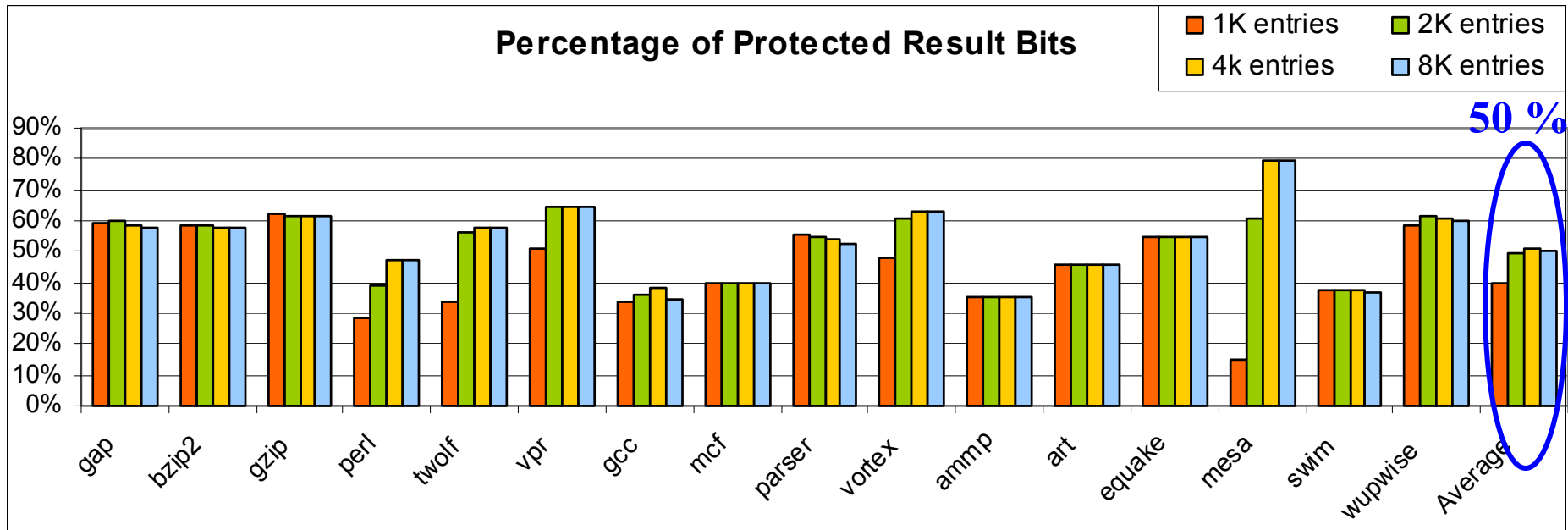


## Experimental Methodology

- Error injection into the Issue Queue and Functional Units
- Compare our approach to:
  - Implicit Redundancy Through Reuse (IRTR) [Gomaa et al., ISCA 2005]
  - Squash on L2-cache miss (SL2) [Weaver et al., ISCA 2004]
  - Squash on Branch misprediction (BR-squash) [Wang et al., DSN 2005]

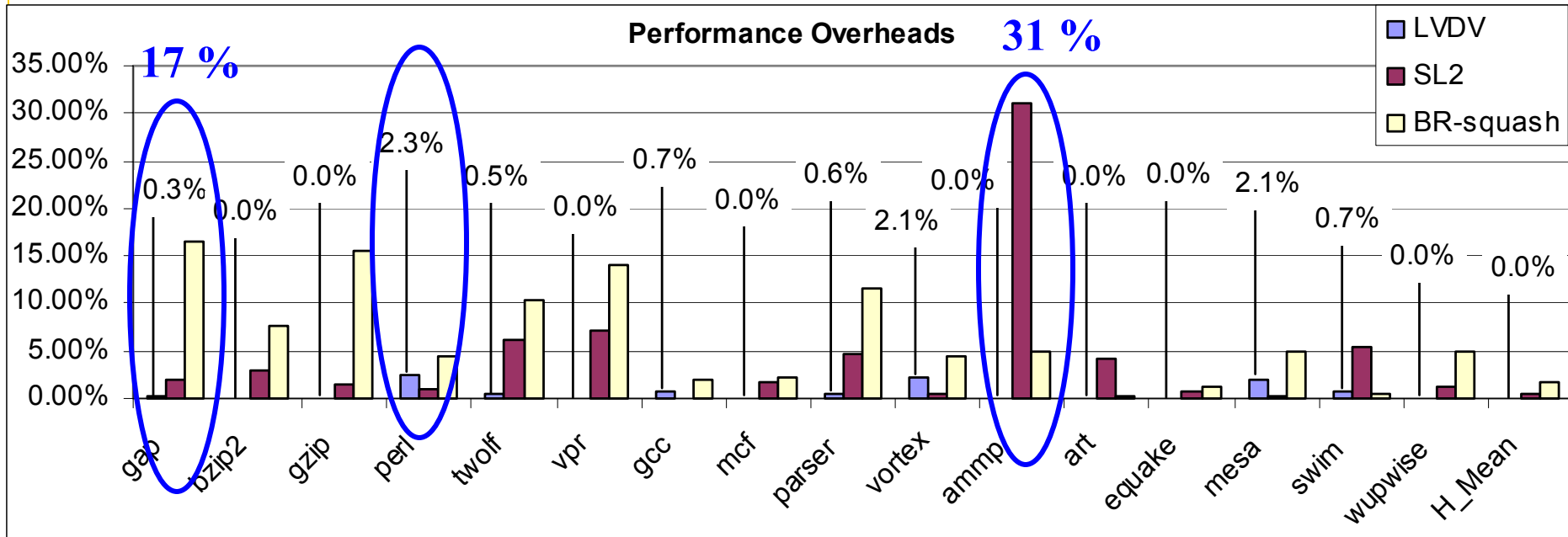


# Strength of the LVDV Locality



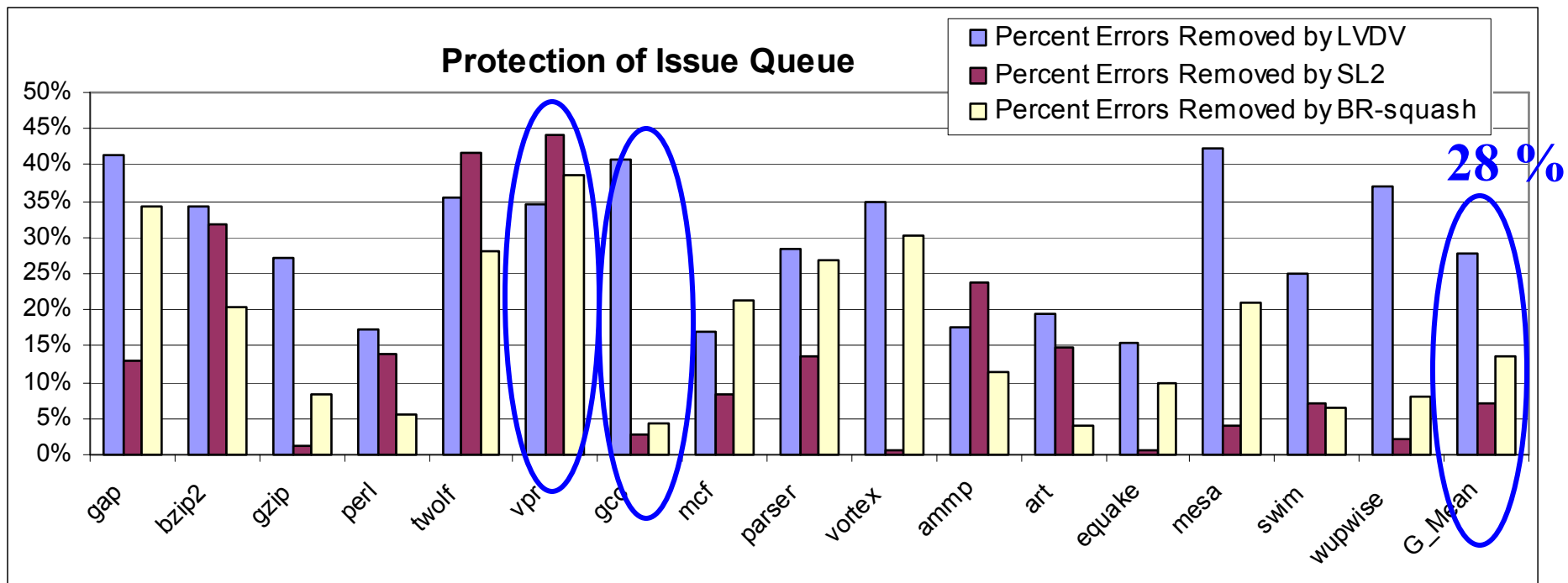


# Performance Overhead





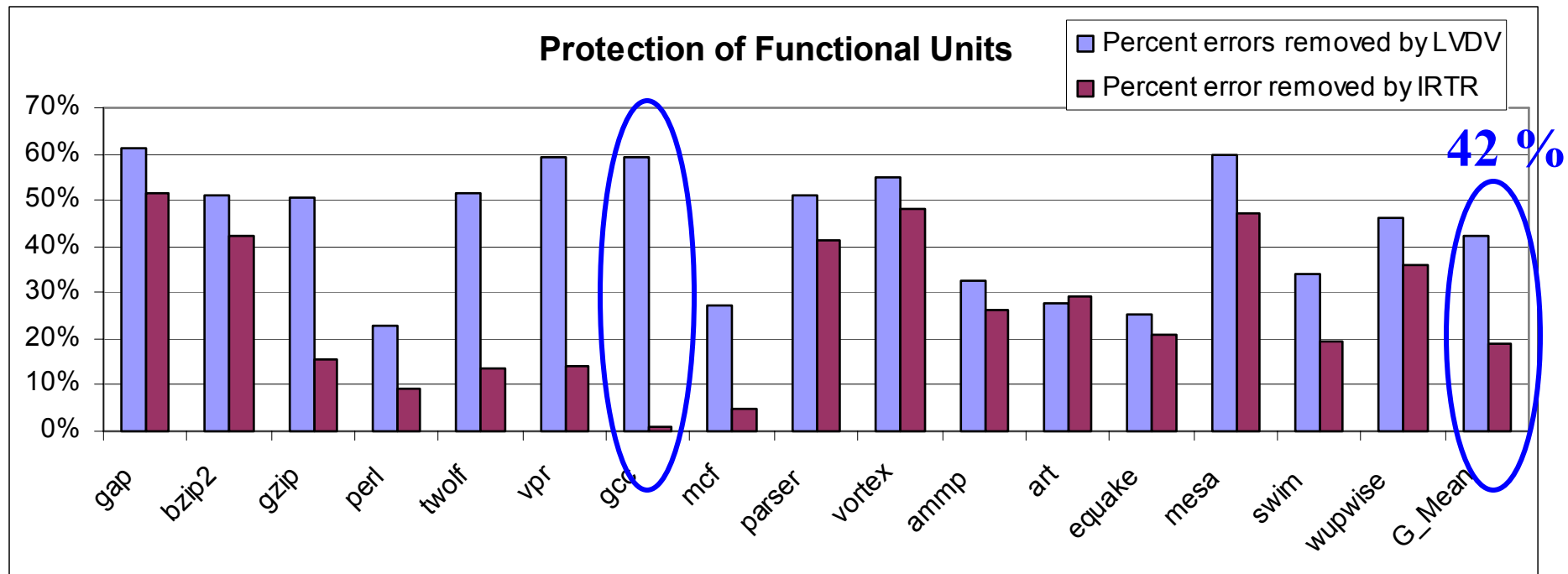
# Soft-Error Protection of Issue Queues



- LVDV removes 28% of critical errors
- MTTF improvement of 39%



# Soft-Error Protection of Functional Units



- LVDV removes up to 61% of critical errors and 42% on average
- MTTF improvement of up to 156% and 72% on average



# Locality-Based Software Bug Detection

- Training phase
  - LVDV table learns invariance information from multiple successful program runs OR from a single long run.
- Bug detection phase
  - LVDV table detects anomalies during the failing run
  - Anomalies are reported as possible root-causes of a software bug
  - Misses in the LVDV table are also signaled as “new-code” anomalies
- We used a 4K entries table and tracked the variance in every store instruction



# Locality-Based Software Bug Detection

- Shown to pinpoint latent program bugs
- Can detect bugs, which do not violate any programming rules
- Our approach can be viewed as a hardware implementation of DIDUCE [Hangal et al, ICSE 2002]
- Hardware offers the following advantages:
  - Performance efficiency
  - Binary compatibility
  - Runtime monitoring



## Case Study

### Incorrect Bounds Checking

```
char newname[MAX];

/* convert the filename */
for(i=0;i<strlen(original);i++){ /*bug*/
    if( isupper( original[i] ) ){
        newname[i]= tolower(original[i]);
        continue;
    }
    newname[i] = original[i]; /*detection*/
}
newname[i] = '\0'; /*detection*/
```

*A buffer overflow in polymorph-0.4.0*



## Case Study Warm-up Phase

```
char newname[MAX];  
  
/* convert the filename */  
for(i=0;i<strlen(original);i++){ /*bug*/  
    if( isupper( original[i] ) ){  
        newname[i]= tolower(original[i]);  
        continue;  
    }  
    newname[i] = original[i]; /*detection*/  
}  
newname[i] = '\0'; /*detection*/
```

```
&newname[3]:    7fffaf33  
&newname[6]:    7fffaf36  
&newname[11]:   7fffaf3b  
&newname[2]:    7fffaf32  
&newname[17]:   7fffaf41  
&newname[9]:    7fffaf39  
&newname[5]:    7fffaf35  
...             ...  
&newname[16]:   7fffaf40
```

*conf: 15 prev\_result: 7fffaf40 encoded variance: 7*



## Case Study Detection Phase

```
char newname[MAX];  
  
/* convert the filename */  
for(i=0;i<strlen(original);i++){ /*bug*/  
    if( isupper( original[i] ) ){  
        newname[i]= tolower(original[i]);  
        continue;  
    }  
    newname[i] = original[i]; /*detection*/  
}  
newname[i] = '\0'; /*detection*/
```

```
&newname[85]: 7fffaf85
```

*conf: 15 encoded variance: 7 variance: 8*



## Summary of Results

- 4K LVDV provides similar bug detection capabilities as software approach DIDUCE
- Also similar number of false-positives as DIDUCE

	Polymorph	bc	Ncompress (input 1)	ncompress (input 2)	gzip
DIDUCE	2	45	1	0	6
4K LVDV	2	54	1	0	6



# Summary

- Soft-errors and software bugs manifest in similar ways during execution
- We can use localities to detect both
- Proposed mechanism protects multiple structures from soft-errors ( 39% and 72% MTTF of IQ and FU)
- Similar bug detection capabilities to DIDUCE (a software based bug detection scheme)