Architectural Methods to Understand Soft Errors/ Process Variations in DSN 2012

Jun YAO

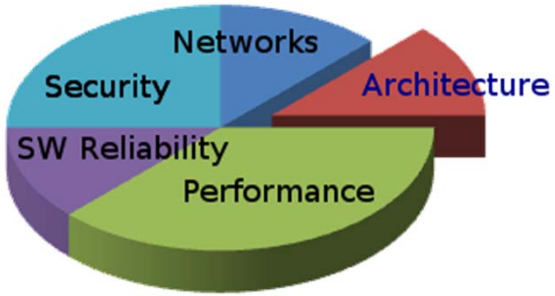
Nara Institute of Science and Technology

yaojun@is.naist.jp

Brief Introduction of DSN'12

- DSN'12
  - The 42nd Annual IEEE/IFIP International Conference on Dependable System and Networks
  - Two symposiums into one conference
    - PDS: Performance and Dependability Symposium
      - Performance, dependability and security;
    - DCCS: Dependable Computing and Communication Systems
      - Dependability and security.

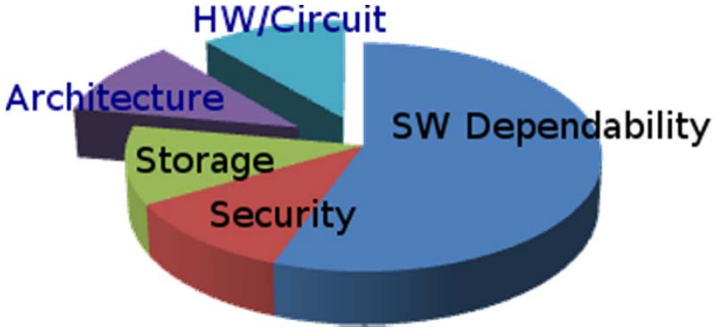
Fields of Papers



- DSN/PCS
  - 24 papers accepted at a rate of 30%.
    - 3 related to processor architecture or lower level (12.5%)
- DSN/DCCS
  - 27 papers accepted at a rate of 17.3%.

ISCA 2012: Rate 18%.

- Image:
  - Far more SW than HW.
  - Security/availability are more preferred.
  - Should try PDS (rejected by DCCS).



Papers of Interests

- Understanding Soft Error Propagation Using Efficient Vulnerability-Driven Fault Injection -- *PDS*
  - Xin Xu and Man-Lap Li@George Washington Univ.
  - doi: 10.1109/DSN.2012.6263923
  - Purpose:
    - Effectively inject error during simulation/validation
    - Understanding the output of error injection

Soft Errors in Microprocessors

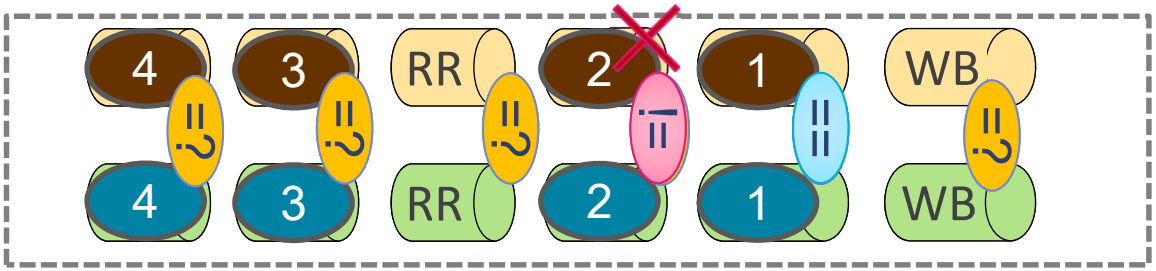
- Causes: particle strikes, radiation…
- Consequences:
  - Abort: System crash or hang, application abnormally exit
  - Silent data corruption (SDC): wrong application outputs when application not abort
  - Masked: Fault is not visible

> 90%

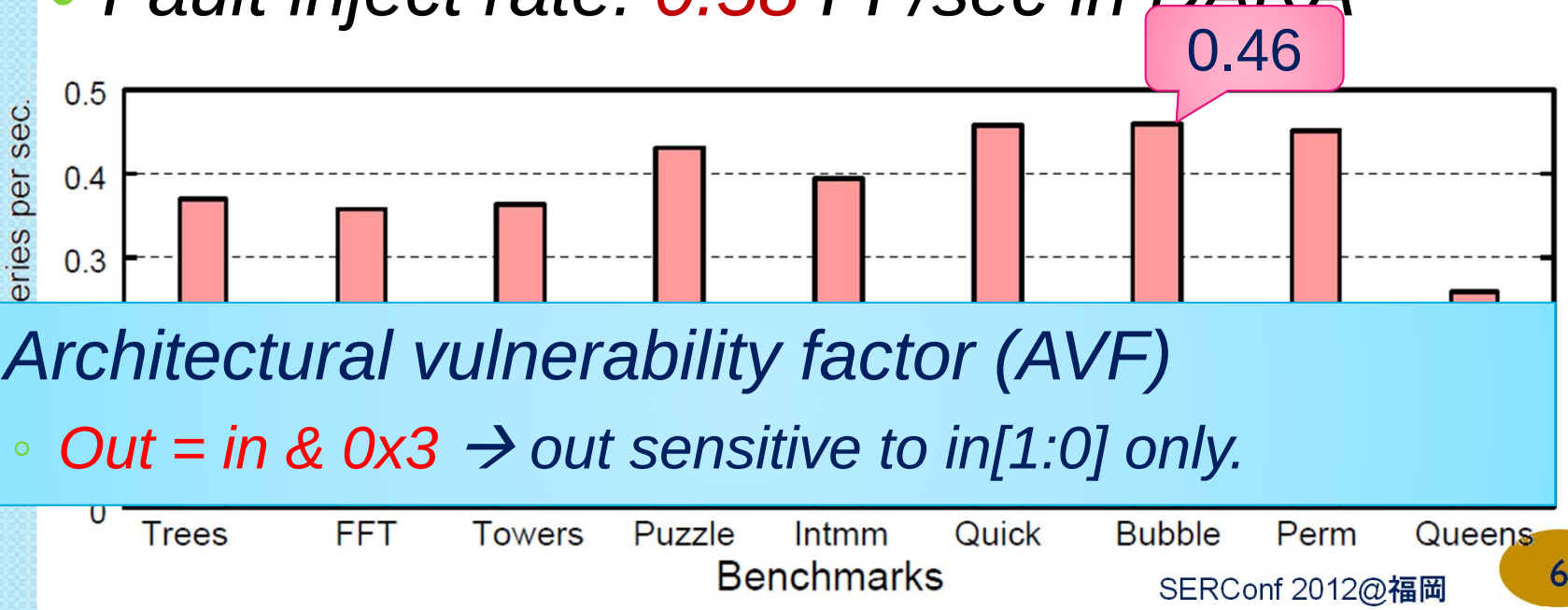
- ✓ Lower the cost for detection & protection
- × Bad for evaluation.

Some data from Yao's research

DMR processor: DARA



- *Alpha source➔*
- *Fault inject rate: 0.58 FF/sec in DARA*



- *Architectural vulnerability factor (AVF)*
  - *Out = in & 0x3 → out sensitive to in[1:0] only.*

Vulnerability-Driven Fault Injection

- Goals: Reduce the error injected on masked values;
  - Same amount of injection get more erroneous result.
- Approach:
  - Guide error injection by vulnerability analysis
- Results:
  - Increases error occurrence by 59%.

VA: get CriticalFault injection space

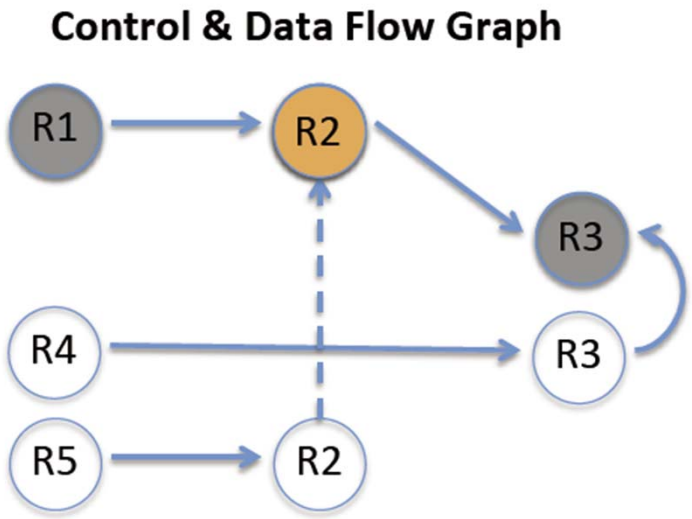
Instruction trace:

Load [R1] ➜ R2;

Add 0x1, R2 ➜ R3;

Move R4 ➜ R3;

Load [R5] ➜ R2;



- First-level dynamically dead (FDD) instruction
  - Above Add instruction;
- Transitively dynamically dead (TDD) instruction
  - Result generated but not consumed.

Remove to get critical fault injection space.

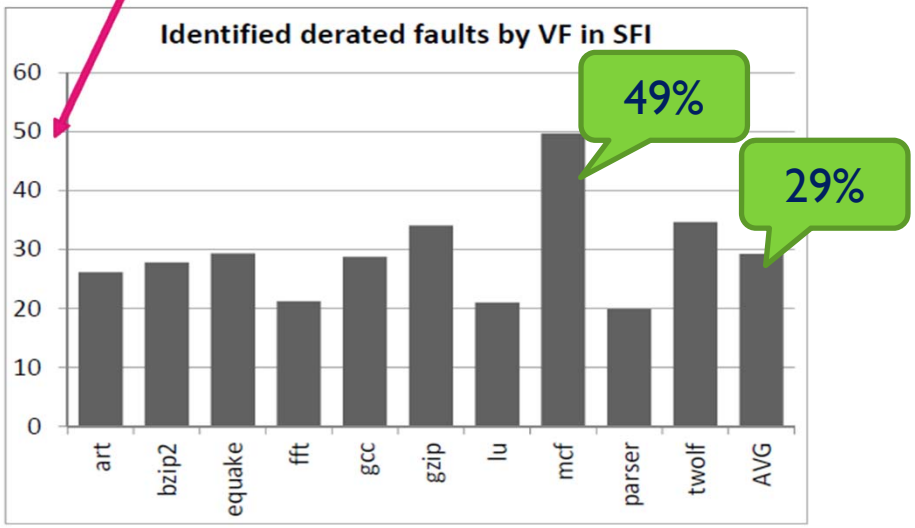
Guided Error Injection Flow

1. Collect instruction trace

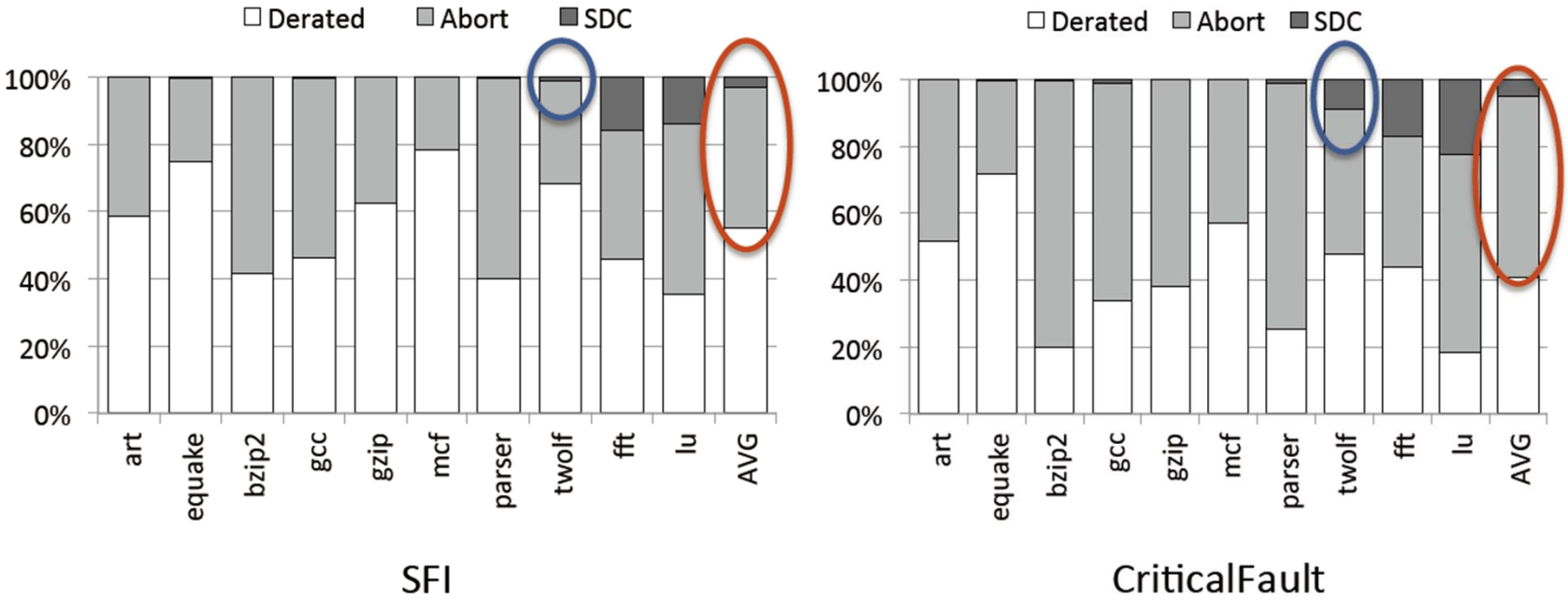
2. Generate injection map (reduced)

3. Simulation: randomly error injection guided by the map.

4.Results analysis (visible error ?).



Overall Error Injection Result



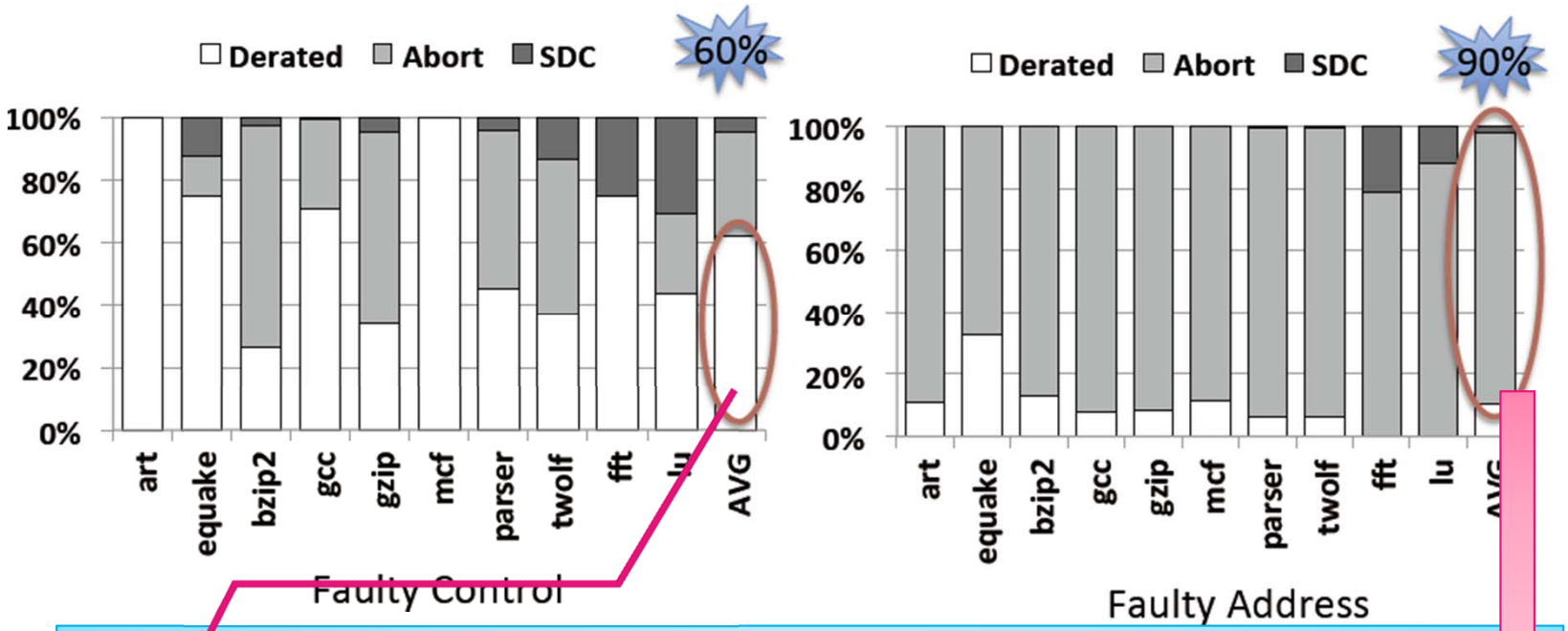
- CriticalFault provides 18% more error occurrence in average
- SDC error increases under guided injection.

More Interesting Results

- Classify injection by types
  - Three categories
    - Faulty control: T= a>0; if (T) goto Loop_exit;
    - Faulty address: LD [R1]➜R2; ST R2➜[R3];
    - Faulty data: a = b + c; etc.

- *Two kinds of explorations:*
  - *How soft error is propagated inside the processor*
  - *How long will it be a problem*

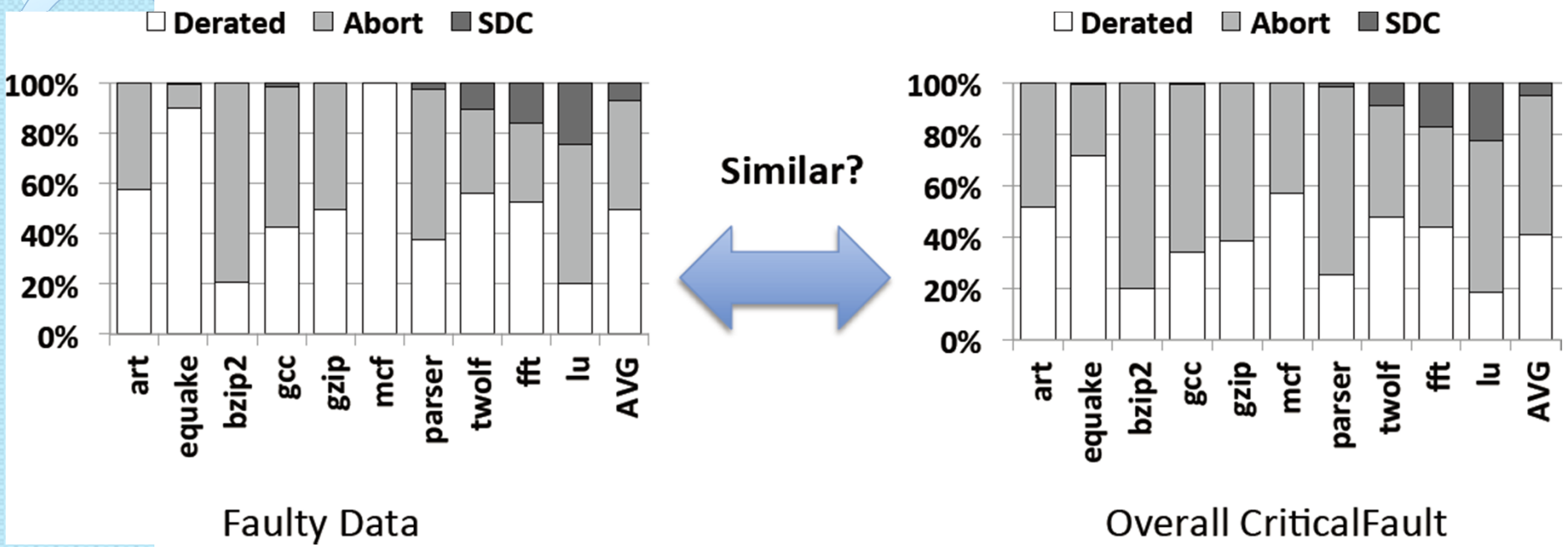
Faulty control/address



- 60% of faulty control is not visible to the final program results. ➜ quite different to my imagination.

- 90% address faults results to ABORT:
  - High necessity to cover address.

Faulty Data

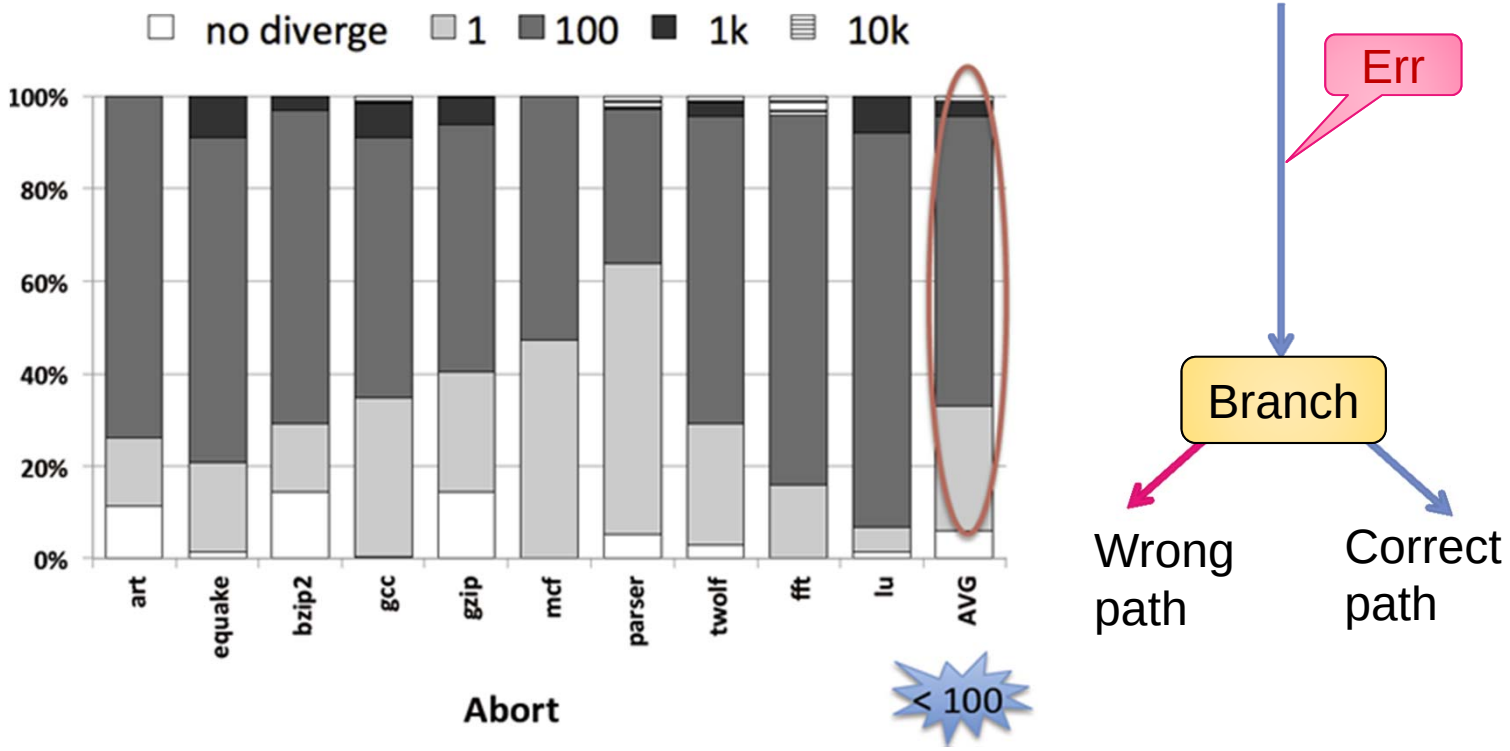


Faulty Data

Overall CriticalFault

- Resemblance of faulty data to all faults
  - Faulty data leads to all possibility cases

Life Time of Injected Error



- Under abort cases, the control path will divert within 100 instructions.

Conclusion

- Give a way to reduce the error injection space
- Show the responses of different instruction types upon error injection

- *My image: balancing cost & behaviors are important*
  - *Cost for redundancy is always high*
  - *But without redundancy, we can not trace error occurrence.*

Papers of Interests

- VARIUS-NTV: A Microarchitectural Model to Capture the Increased Sensitivity of Manycores to Process Variations at Near-Threshold Voltages -- *DCCS*
  - *Ulya R. Karpuzcu, Krishna B. Kolluru, Nam Sung Kim, and Josep Torrellas@UIUC*
  - doi: 10.1109/DSN.2012.6263951
  - Purpose: Modeling Process Variation
    - Estimate NTV in many core architecture

Approaches

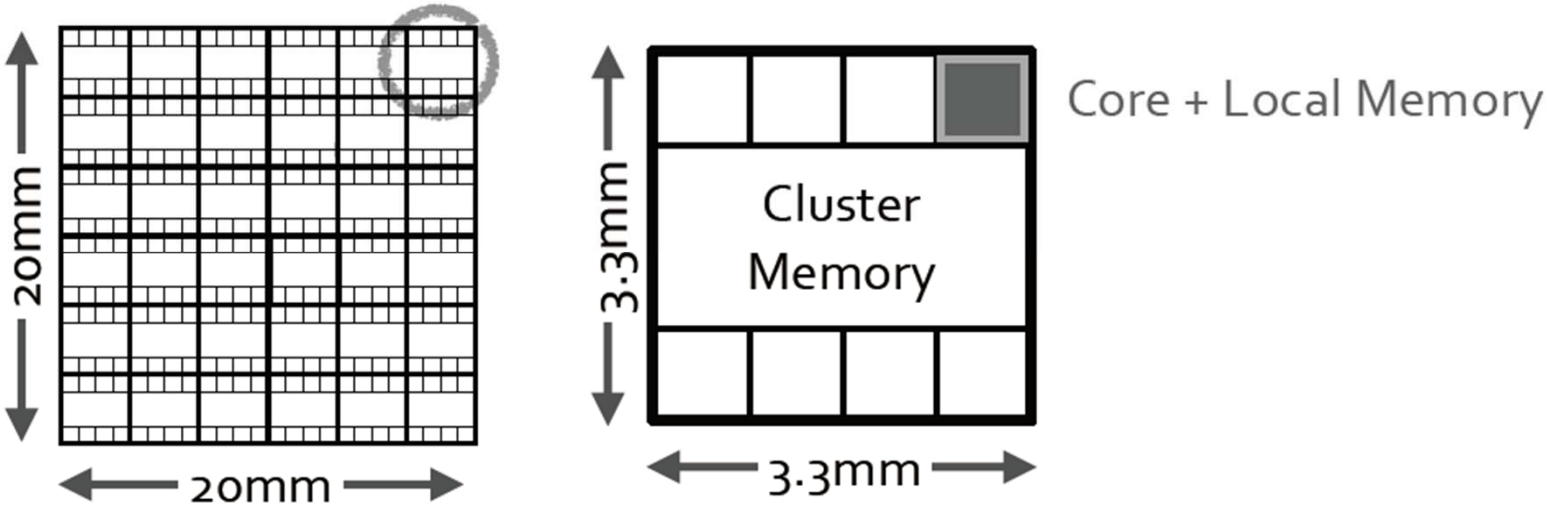
- Extends existing VARIUS, adds *NTV*

Gate Delay Model	EKV Based
SRAM Cell Architecture	8T
SRAM Failure Modes	Hold Write Stability Write Timing Read Timing
Impact of Leakage	✓

- Download at:
  - http://iacoma.cs.uiuc.edu/varius/ntv/variusNTV.html

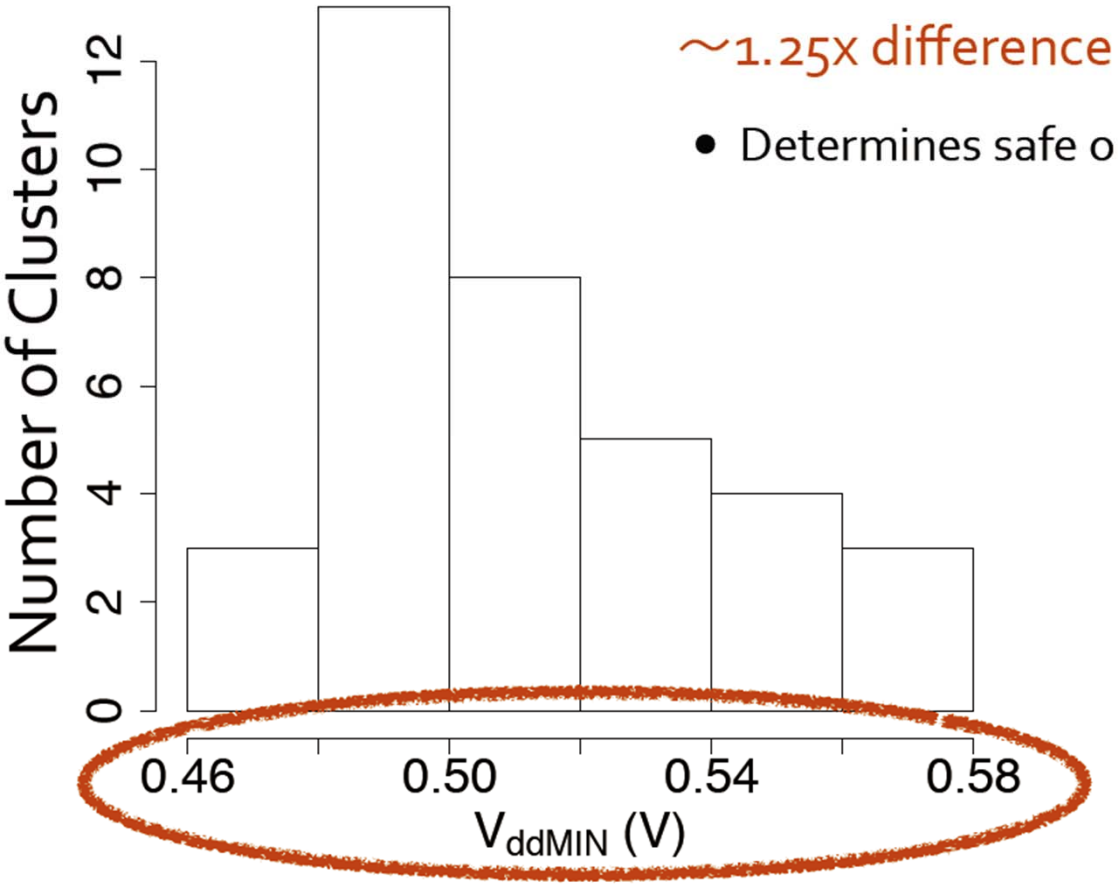
Evaluation Setup

- 288 core chip
  - 36 clusters, 8 cores per cluster
  - Core: single issue in-order
- 11nm process



Results

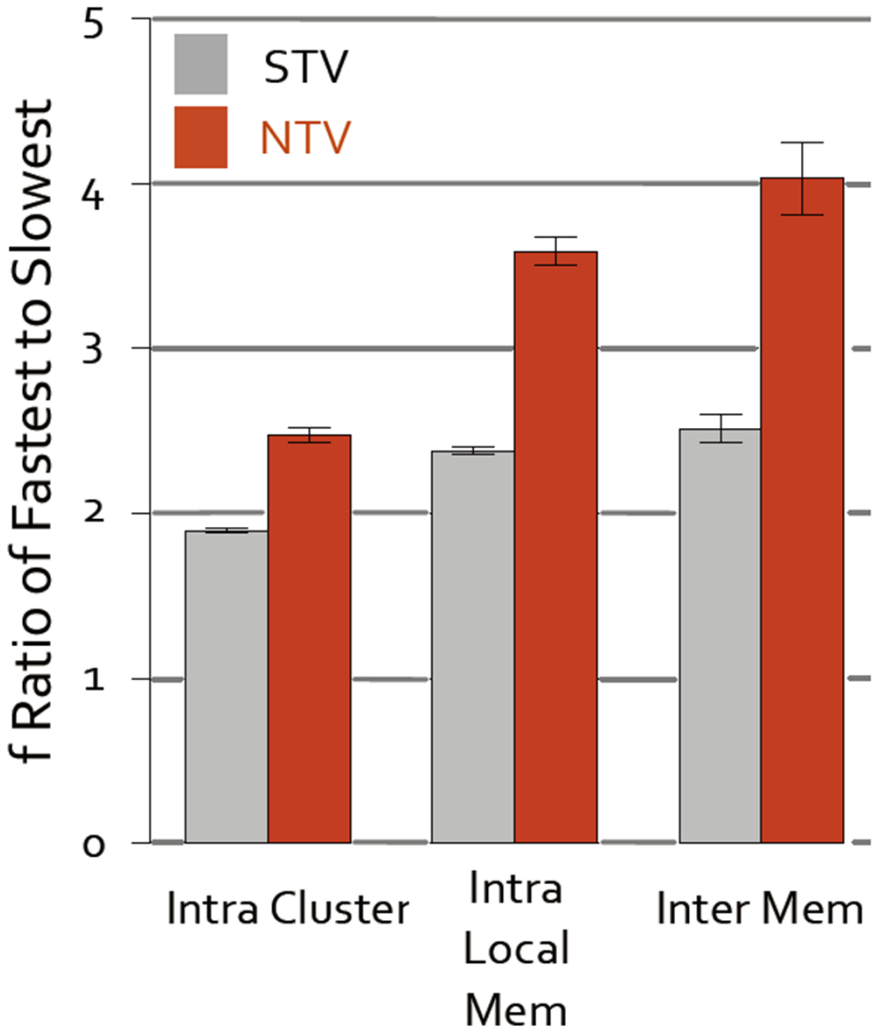
- Variations in Vddmin



~1.25x difference across clusters

- Determines safe operating Vdd

Variation of Frequency



- Sub-threshold zone
  - 2.3x
- NTV zone
  - 3.7x difference

- *Please download*
- *and try it.*

The End