The Case for Run-Time Error Checking

Todd Austin
Advanced Computer Architecture Lab
University of Michigan

Classic Wins in Run-Time Error Correction
Run-Time Error Correction

Benefits of Run-Time Error Correction

- Reduce design complexity/cost
  - E.g., checker covers hard to find bugs, reduces time-to-market
  - E.g., checker lends itself to full formal verification
- Reduce manufacturing complexity/cost
  - E.g., fully-testable checker covers defects missed in checked components
  - E.g., on-chip checkers can be used as a high-B/W tester
- Optimize a design by eliminating fault-avoidance margins/complexity
  - E.g., Razor circuit operation at subcritical voltages
  - E.g., iA32 checker covers partial memory forwards thru virtual aliases
- Correct run-time upsets
  - E.g., cover SER events
  - E.g., design for unlikely noise events (rather than "possible" noise events)
Deployment Challenges

- Designer mind-set
  - “This is a step backwards to go forward.”
  - “This is a `sloppy' approach.”
- Remedies
  - Growing GSRC mindshare
  - Generate value-added applications
  - Define desirable “baby steps” to ultimate goals

Functional Correction: DIVA Checker Processor

- The fatalist's approach to microprocessor verification!
- Core technology: dynamic verification
  - Simple (and correct) checker processor verifies all results before retirement
  - Reduces the burden of correctness on the core processor design
  - Checker can be simple by relying on core for branch/address predictions
- Fundamentally changes the design of a complex microprocessor
  - Beta release processors
  - Low-cost SER protection
**Timing Correction: Razor Low-Power Pipeline**

- *In-situ* error detection and correction
  - Delayed shadow latch secures a "second opinion" on all stage computation
  - Detected errors corrected using microarchitecture speculation recovery mechanism
  - Tune processor voltage based on error rate
    - Eliminate process, temperature, and safety margins
    - Purposely run below critical operation voltage to capture *data margins*

---

**How Can the Simple Checker Keep Up?**

- Slipstream reduces power requirements of trailing car
- Checker processor executes inside core processor’s slipstream
  - Fast moving air ⇒ branch/value predictions and cache prefetches
  - Core processor slipstream reduces complexity requirements of checker
  - Checker rarely sees branch mispredictions, data hazards, or cache misses
How Can the Simple Checker Keep Up?

- Slipstream reduces power requirements of trailing car
- Checker processor executes inside core processor’s slipstream
  - fast moving air \(\Rightarrow\) branch/value predictions and cache prefetches
  - Core processor slipstream reduces complexity requirements of checker
  - Checker rarely sees branch mispredictions, data hazards, or cache misses