

Ultra Low-Cost Defect Protection for Microprocessor Pipelines



Smitha Shyam Kypros Constantinides Sujay Phadke
Valeria Bertacco Todd Austin

Advanced Computer Architecture Lab
University of Michigan



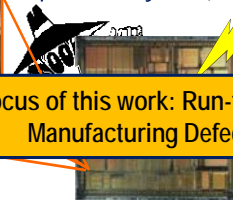
EECS 573
January 29, 2007

Key Reliability Threats

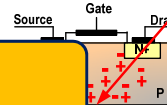
Run-Time Defects
(Wire break-down and transistor wear-out)



H/W and S/W Design Errors
(Bugs are expensive and expose security holes)



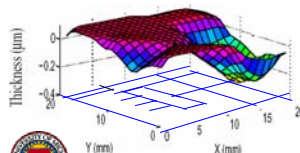
Transient Faults due to Cosmic Rays & Alpha Particles
(Increase exponentially with number of devices on chip)



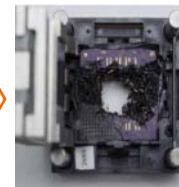
Focus of this work: Run-time and Manufacturing Defects

Parametric Variability
(Uncertainty in device and environment)

Intra-die variations in ILD thickness



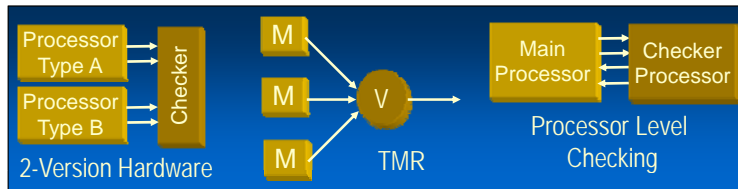
Manufacturing Defects That Escape Testing
(Inefficient Burn-in Testing)



Traditional Defect-Tolerant Techniques

◆ Used at high-end life-critical systems

- N-Version Hardware
- Triple Modular Redundancy (voting scheme)
- Microprocessor Checkers



◆ Utilize redundant hardware to validate computation

- Results in very high area cost
- Very costly to employ for mainstream systems



January 29, 2007

Goal: BulletProof Pipeline

◆ Area Cost

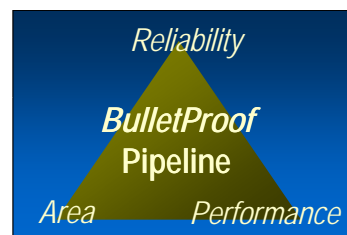
- Ultra low-cost solution

◆ Provided Reliability

- Support recovery from first defect

◆ Performance

- After recovery the system still operates in degraded performance mode



EECS 573
January 29, 2007

Approach: BulletProof Pipeline

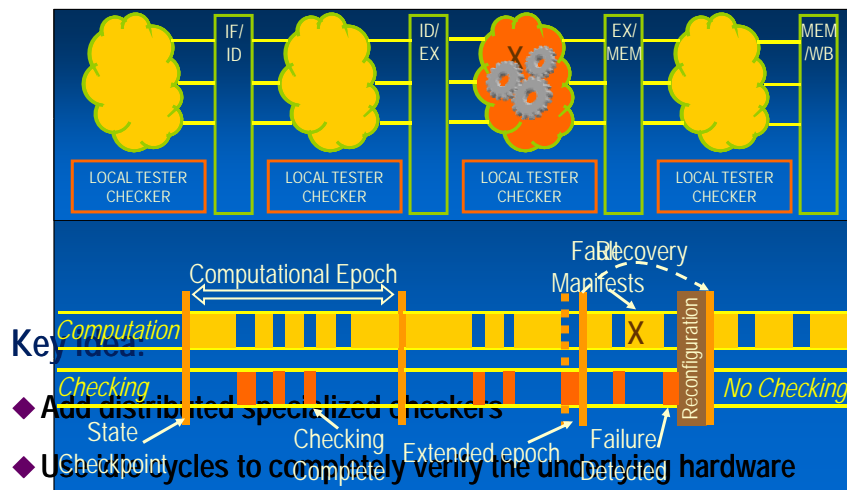
- ◆ Employ microarchitectural checkpointing to provide a *computational epoch*
- ◆ *Computational Epoch*: a protected period of computation over which the underlying hardware is checked
- ◆ Use on-line distributed testing techniques to verify the hardware is free of defects, on idle cycles
- ◆ If a component is defective disable it, rollback state, and continue operation under a degraded performance mode on remaining resources

For inexpensive defect protection, don't check computation, Instead... Validate H/W is free of defects, otherwise, rollback and recover.



EECS 573
January 29, 2007

Distributed Testing and Recovery



EECS 573
January 29, 2007

Micro-Architectural Checkpointing

◆ A mechanism to create coarse-grained epochs of execution

- Augment each cache block with a *Volatile* bit to indicate speculative state
- Backup Register File: single-port SRAM (simpler and smaller than regular RF)



◆ A computational epoch must end when:

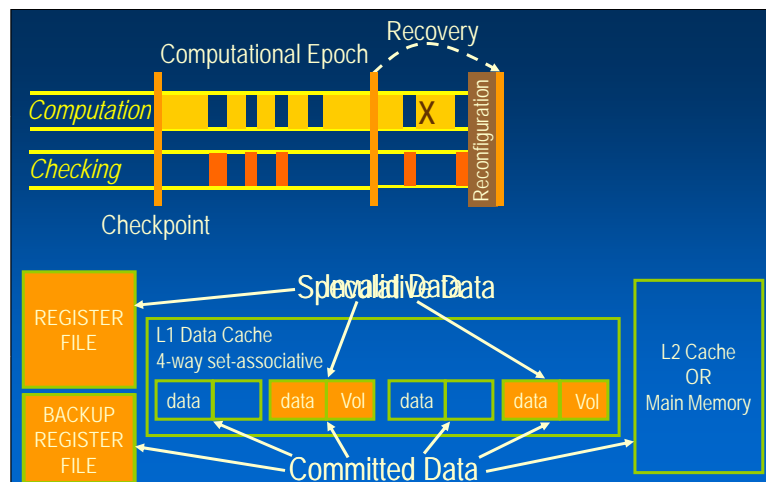
- All cache blocks in a set are volatile OR an I/O operation is requested

◆ Average epoch size is in the order of 10,000+ of instructions



January 29, 2007

Micro-Architectural Checkpointing

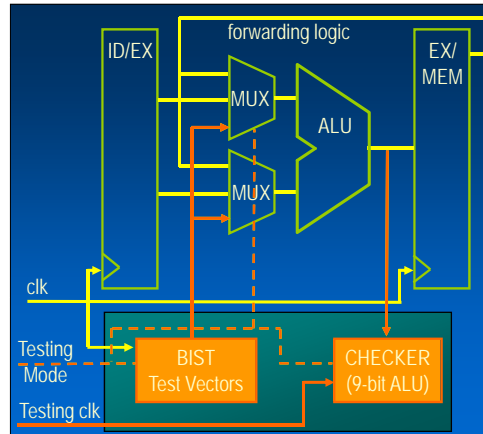


EECS 573
January 29, 2007

Specialized Distributed Online Testing/Checking

Tester/Checker for the ALU/Address Generation Unit

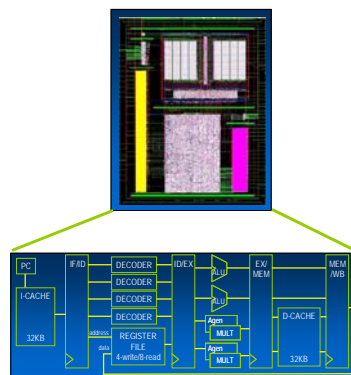
- On idle cycles the ALU enters into testing mode
- Built-In Self-Test vectors are sent to ALU
- Output verified by a 9-bit mini-ALU checker
- 4 cycles to fully verify the ALU
- Other checkers covered in paper



EECS 573
January 29, 2007

Experimental Methodology - Baseline Architecture

- ◆ **Baseline Architecture:**
 - 5-stage 4-wide VLIW architecture, 32KB I-Cache, 32KB D-Cache
 - Embedded designs: Need high reliability with high cost sensitivity
- ◆ **Circuit-Level Evaluation:**
 - Prototype with a physical layout (TSMC 0.18um)
 - Accurate area overhead estimations
 - Accurate fault coverage area estimations
- ◆ **Architecture-Level Evaluation:**
 - Trimaran toolset & Dinero IV cache simulator
 - Average computational epoch size
 - Performance while in graceful degradation
- ◆ **Benchmarks:**
 - SPECINT2000, MediaBench, MiBench



January 29, 2007

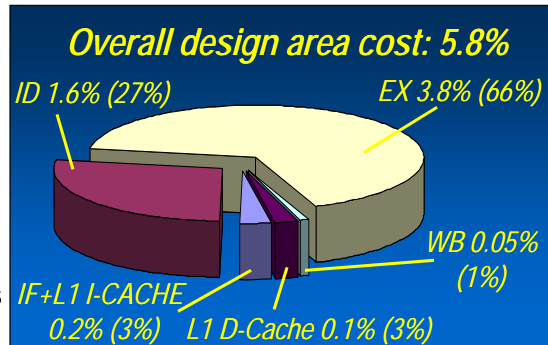
Area Overhead Summary

- ◆ Overhead calculated using a physical-level prototype
 - Place & routed synthesized Verilog description of the design

- ◆ EX stage dominates area cost contribution
 - Functional unit checkers
 - Test vectors

- ◆ Next is ID stage
 - Decoder checkers
 - Test vectors
 - Backup register file

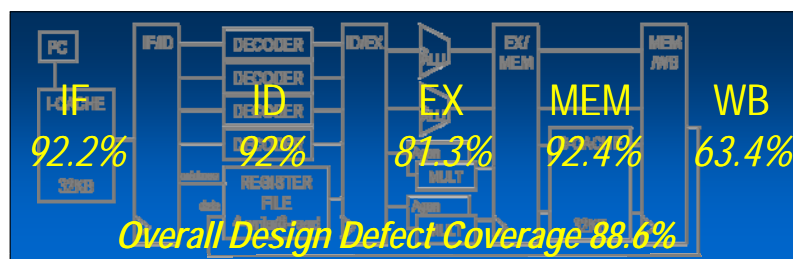
- ◆ The rest is:
 - Cache parity/volatile bits
 - Testing logic



EECS 573
January 29, 2007

Design Defect Coverage

- ◆ **Defect Coverage:** total area of the design in which a defect can be detected and corrected



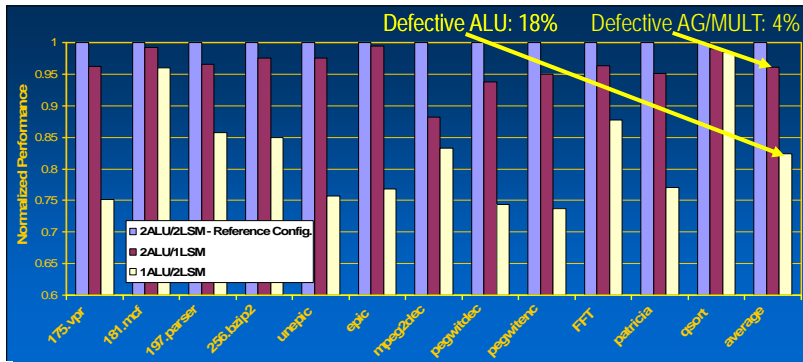
- ◆ The unprotected area of the design mainly consists:
 - Resources that do not exhibit inherent redundancy
 - E.g., Interconnect (i.e., buses connecting the components) and control logic



EECS 573
January 29, 2007

Performance Under Degraded Mode Execution

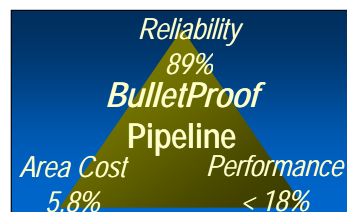
- ◆ The system recovers from a defect by disabling the defective component
- ◆ Losing an ALU results in average 18% performance degradation
- ◆ Losing an Addr. Gen/MULT unit results in average 4% perf. degradation



January 29, 2007

Conclusions

- ◆ Presented the *BulletProof* pipeline
 - First ultra-low cost defect protection mechanism for microprocessors
 - Propose the combination of on-line distributed testing with microarchitectural checkpointing for low-cost defect protection
- ◆ Implemented a physical-level prototype of the technique
 - *Area cost*: 5.8%
 - *Reliability*: 89%
(coverage for first defect)
 - *Performance loss*: 18%
(after graceful degradation)



EECS 573
January 29, 2007

Discussion Points

- ◆ How useful is single defect coverage?
- ◆ Is the measured design coverage “good enough”?
- ◆ Does it make sense to build in defect coverage without support for soft-errors?



EECS 573
January 29, 2007