SWAT: Hardware Reliability through Software Anomaly Treatment

Manlap Li, Pradeep Ramachandran, Swarup Sahoo, Siva Kumar Sastry Hari,
Sarita Adve, Vikram Adve, Yuanyuan Zhou
University of Illinois, Urbana-Champaign

Resilient Theme, Task # 1.2.2.5

GSRC Annual Symposium
Sep 29-30, 2008

Motivation

CMOS scaling ⇒ ↑ device failure ⇒ in-field hardware failure

Need for on-the-fly detection, diagnosis and recovery/repair

Key Observations

Handle only those h/w faults that propagate to s/w

Optimize common fault-free cases

Strategy

Watch for software anomaly (symptoms)

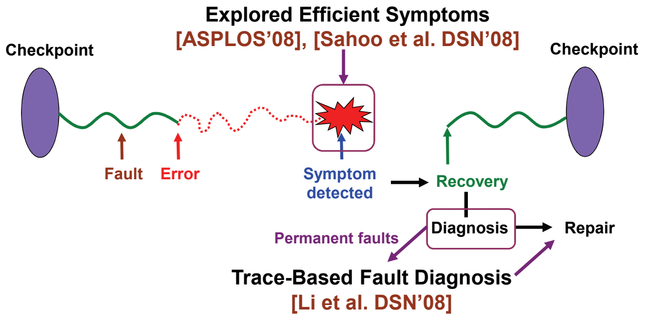
Zero to low overhead *"always-on"* monitors

Diagnosis after detection

May incur high overhead, but *rarely invoked*

Checkpoint/replay based recovery

SWAT: SoftWare Anomaly Treatment



Ongoing work

Recovery

Hybrid hardware/software checkpointing techniques

Hardware handles most cases

Hybrid technique handles (few) long detection latencies

SWAT on Multicore Systems

Understanding fault propagation to fault-free core

~20% of faults corrupt fault-free core execution

New symptoms: No Forward Progress, CPU Panic

iSWAT [Sahoo et al. DSN'08]

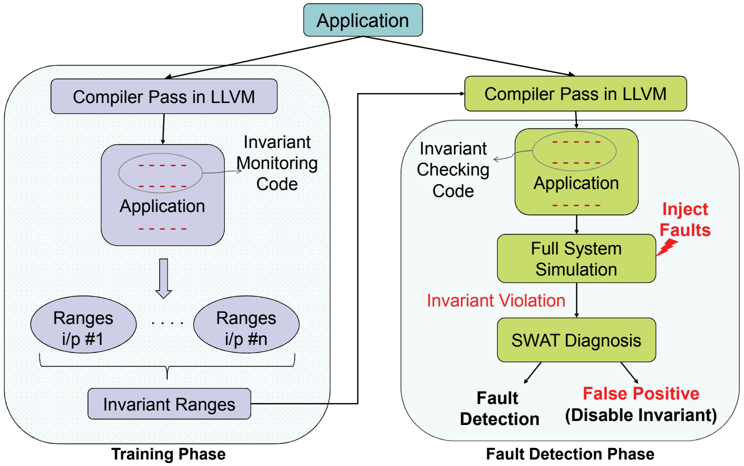
Detection using likely program invariants

Observation:

Undetected faults in SWAT mostly corrupt data values

Strategy:

- Check data values with range-based likely invariants (*MIN≤ value≤ MAX)*
  - ◊ Instrument the binary with invariant checking code
- Exploit SWAT diagnosis to identify false positive invariants at runtime
  - ◊ Disable false positive invariants, limit diagnosis overhead
    - ▪ Maximum number of rollback <= number of static false positives



Key Results:

- **False Positive rate is less than 5% with 12 training inputs**
- **Undetected faults reduced by 28% and SDCs reduce by 74%**
- **Low run-time overhead (5% on x86, 14% on UltraSPARC-IIIi)**

Diagnosis [Li et al. DSN'08]

Symptom-based detection ⇒ distinguish software bugs, transients, permanent faults

Permanent fault needs repair ⇒ disabling core wasteful ⇒ reconfigure faulty *μarch units*

Opportunities

- Exploit checkpoint/replay on multicore system
- Replay faulty and fault-free execution, compare
  - ⇒ **Synthesized DMR** for diagnosis

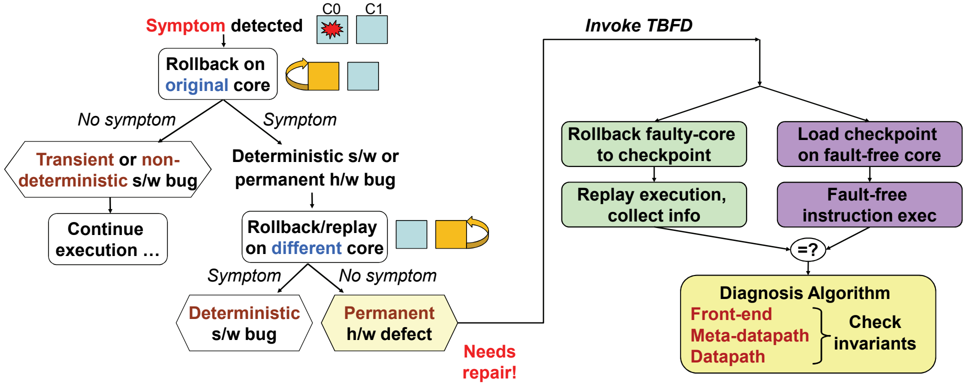
Trace-Based Fault Diagnosis (TBFD) for μarch unit

Key Ideas

- Compare instruction trace of faulty vs. fault-free execution
- Divergence ⇒ faulty hardware used ⇒ diagnosis clues

Advantages

- Works with other detection techniques
- Supports different repair granularity


C0
C1
Symptom detected
Rollback on original core
No symptom
Symptom
Transient or non-deterministic s/w bug
Deterministic s/w or permanent h/w bug
Continue execution …
Rollback/replay on different core
Symptom
No symptom
Deterministic s/w bug
Permanent h/w defect
Needs repair!
Invoke TBFD
Rollback faulty-core to checkpoint
Load checkpoint on fault-free core
Replay execution, collect info
Fault-free instruction exec
=?
Diagnosis Algorithm
Front-end
Meta-datapath
Datapath
Check invariants

Key Results:

- **98% of detected faults can be diagnosed**
- **~90% diagnosed to exact array entry/unit**
- **Logical-physical register mapping (meta-data path) faults require sophisticated algorithm**