ASeD: Availability, Security, and Debugging Support using Transactional Memory

JaeWoong Chung, Woongki Baek, Nathan G. Bronson, Jiwon Seo, Christos Kozyrakis, Kunle Olukotun

Stanford University

Motivation

- Transactional Memory (TM)
  - Simplifies parallel programming using atomic blocks
  - Easy to use & high performance
- TM systems should provide ACI features
  - Atomicity: rollback to a safe system state
  - Consistency: guarantee system-level invariants
  - Isolation: limit the propagation of side effects
- Key insights: ACI of TM can also be used for ASeD
  - Availability
  - Security
  - Debugging

Contributions

- Investigation of the feasibility of ASeD on HTMs
  - Demonstrate the great synergy between ASeD and ACI
- Implementation of the ASeD on HTM by addressing:
  - Tightly coupled ACI components in HTM
  - Enhancing performance-oriented ACI in HTM for ASeD
  - Simultaneous use of ACI for concurrency and ASeD
- Quantitative evaluation of the proposed HTM with ASeD
  - Overall, achieving ASeD with small runtime overhead

ASeD Design Philosophy

- Tool vs. Solution
  - Providing a tool with key primitives for flexibility
  - Reducing HW acceleration for the common case
- Integration vs. Versatility
  - Proposing an integrated design instead of a collection of separate HW extensions
- Cost-efficiency vs. Performance
  - Avoiding additional HW just to accelerate a single feature
  - Maximizing HW resources between ASeD & HTM

ASeD on Transactional Memory

- Implementing the ASeD on HTM using similarity of ACI
  - Reduces HW costs of ASeD support
  - An integrated system of TM with the advantages of ASeD

Availability Features

- ASeD addresses both permanent & transient faults
  - Permanent: loss in cores or caches, etc.
  - Transient: packet loss, logic errors, etc.
- Availability primitives
  - Global checkpoint: system-wide state
  - AI (atomic/isolated) regions: thread-specific state
- Use case
  - Permanent faults
    - Global checkpoints are periodically taken
    - Upon faults, roll-back to the latest global checkpoint
  - Transient faults
    - Code fragments are enclosed by AI regions
    - Upon faults, just roll-back the faulty thread
    - Significantly reduced MTTR

Security Features

- Security primitives
  - Fine grained readWrite barriers
  - Accessing a specific address is detected & notified
  - Isolated execution
    - Similar to the AI regions
- Use case
  - Buffer overflow detection
    - Mark the address of canaries with write barriers
  - Overwrite a canary is detected by the ASeD
    - Significantly lower overhead than SW canaries

Debugging Features

- Debugging primitives
  - Global checkpoint
  - Fine grained readWrite barriers
- Use case
  - Scalable watchpoints
    - Provides arbitrary # of watchpoints using RW barriers
    - Negligible performance impact
    - Supports coarse-grain watchpoints with the runtime
  - Parallel bookmark & step-back are also supported

ASeD vs. HTM

- Comparison of ASeD and HTM
  - ASeD addresses both permanent & transient faults
  - HTM addresses only permanent faults
  - ASeD is designed for debugging
  - HTM is designed for isolation

Availability Experiments

- Normalized execution of 8 apps with 4 versions
  - Base: without checkpoints & faults
  - GCP: global checkpoint every 50K cycles, no faults
  - GR: same as GCP with fault injection/1M cycles
  - LR: same as GR with local recovery using AIs
- Summary of results
  - GCP overhead: < 3.5%, GR overhead: 20-30%
  - LR reduces the overhead of GR (Equake: false conflicts)

ASeD – HTM

- Major difference is global checkpoint (system-wide)
  - Periodic interrupts stop/complete all consistency
  - Flush all dirty cache lines to memory
  - HTM mechanisms are used to checkpoint reg. state
  - When an error is detected
    - All cores/caches are reset
    - All logs are applied in reverse order
  - Register checkpoint is restored and system resumes

Feature Decomposition

- Summary of results
  - ASeD: 0.2% – 0.4% overhead
  - StackGuard: 3.1% – 6.6% overhead

Security Experiments

- Overhead of ASeD is significantly lower than StackGuard
  - StackGuard: a number of inst’s to set/check canaries
  - ASeD: only two inst’s at prologue & epilogue of functions

Conclusions

- We proposed & evaluated the ASeD on top of HTM
  - Availability using global & local checkpoints
  - Security using RW barriers & isolated executions
  - Debugging: using global checkpoints
- Overall, enhanced HTM with ASeD with minimal HW costs