Dynamic Verification of Sequential Consistency

Paper by Albert Meixner and Daniel J. Sorin

Presented by: Tom Marmaduke
Design Goals

• Dynamic verification of the consistency model (focused on Sequential Consistency)
• Detect errors that conflict with SC and trigger system recovery
• Errors detected:
  – Memory corruption in caches and memory
  – Cache and memory controllers
    • Faults cause incorrect state
  – Interconnection Network
    • Dropped and corrupted messages
DVSC-Direct

• Send Inform messages for loads and stores
  – Indexed by time <major, minor, ProclID>
• Verification Window Buffer allows processors to see inform messages in program order
• Verification Memory stores blocks, updates based on inform messages
  – On Load-Inform, check if matches Verification Memory, fault if it doesn’t
DVSC-Indirect Sub-Invariants

• Fact: Load gets last store, or the data that was sent at beginning of epoch
• Lemma 1: Exclusive epochs are only epochs at the time
• Lemma 2: Processors perform loads/stores during proper epoch
• Lemma 3: Correct values are passed between processors
DVSC-Indirect

• Use DIVA to check Fact 1 and Lemmas 2 and 3
• Cache Epoch Table
  – Store epoch type, start time (16-bit), hashed starting block value, DataReadyBit
  – Error Correcting Code in cache to prevent corruption
  – Checks state on load/store to verify Lemma 2
• Send Epoch-Inform message at end of epoch to block’s home memory
  – Data address, time and block value of epoch start, time and block value of epoch end
DVSC-Indirect

• Memory Epoch Table
  – Stores Epoch-Inform messages in start time order
    • Stores end time of last Shared epoch, end time of last Exclusive epoch, data value at end of last Exclusive epoch
  – Processes Epoch-Inform messages
    • Check epoch times for overlap, update MET times
    • Check start data in Epoch-Inform message matches MET’s last value, update MET if necessary
Evaluation

• Error Coverage:
  – No false positives when epochs were processed out of order

• Performance:
  – DVSC normalized runtime always under 1.2

• Bandwidth:
  – A lot more network traffic, but messages are small so normalized bandwidth increase is not too high
Evaluation continued

• Showed how different logical time lengths affected errors, why not performance?
• What happens with more nodes? Less nodes? Would it work better/worse on different network schemes?
Questions

• Is this extra level of fault protection worth the extra bandwidth and hardware?
• Would it scale well for different numbers of nodes? Could the bandwidth overhead cause problems?