Reactive Synchronization Algorithms for Multiprocessors

Beng-Hong Lim and Anant Agarwal

Margoob Mohiyuddin

Outline

• Motivation
• Algorithms
  – Reactive spin lock
  – Reactive fetch-and-op
• Results
  – Locks: test-and-test&set vs. queue
  – Fetch-and-op: lock vs. software combining tree
  – Shared memory vs. message passing

Motivation

• Passive algorithms
  – Choice of protocol fixed ➞ optimized for certain contention/concurrency levels

Reactive Algorithm: Components

• Protocol selection algorithm
  – Which protocol to use for synchronization operation?
    • test&set vs. queuing
      – Low contention ➞ test-set
      – High contention ➞ queuing

• Waiting algorithm
  – What do we do when waiting for synchronization delays?
    • Spinning vs. blocking
    • This paper spins
Challenges

• Protocol selection algorithm
  – What is the current protocol?
    • For each synchronization operation \( \rightarrow \) efficient
    • Multiple processes may try to synchronize at the same time
  – How to change protocols?
    • Correctly update state
    • Frequent protocol changes \( \rightarrow \) efficient
  – When to change protocols?
    • Crossover point architecture dependent \( \rightarrow \) tuning needed

• Waiting algorithm
  – Local operation
    • Different processes can use different waiting algorithms

Generalizing: Required Properties

1. Each protocol has a unique consensus object
   – Atomic access
   – Valid or invalid
2. A process can complete protocol execution after accessing consensus object
3. Protocol state modified only by a process with atomic access to a consensus object

• Allows processes to safely execute an invalid protocol

• Changing protocol \( A \rightarrow B \):
  • Acquire atomic access to B’s consensus object
  • Invalidate A’s consensus object
  • Update state of B to reflect current state of synch. Operation
  • Change mode variable
  • Validate B’s consensus object and release it

Reactive Spin Lock

• Mode variable, test-and-test&set lock, MCS queue lock
• What is the current protocol?
  – Mode variable: TTS \( \rightarrow \) test-and-test&set, QUEUE \( \rightarrow \) queue
    • Read-cached and infrequent mode changes \( \rightarrow \) efficient
  – At most one lock free
    • Protocol change between mode variable access and lock access \( \rightarrow \) processes executing wrong protocol retry
• How to change protocols?
  – Done by lock holder
    • TTS\( \rightarrow \)QUEUE \( \rightarrow \) update mode, release queue lock
    • QUEUE\( \rightarrow \)TTS \( \rightarrow \) update mode, signal queue waiters to retry, release test-and-test&set
    • Ensures at most lock free
• When to change protocols?
  – \# failed test-and-test&set attempts > threshold \( \rightarrow \) TTS\( \rightarrow \)QUEUE
  – \# consecutive lock acquisitions when queue empty > threshold \( \rightarrow \) QUEUE\( \rightarrow \)TTS

Reactive Fetch-and-Op

• test-and-test&set lock, queue lock, software combining tree
  – Consensus object for combining tree is the lock for the root
• What is the current protocol?
  – Mode variable
  – At most one protocols’ consensus object valid
    • Invalid lock access \( \rightarrow \) retry
    – Combining tree \( \rightarrow \) backtrack tree traversal and notify waiting processes to retry
• How to change protocol?
  – Lock holder updates state of target protocol to current value of fetch-and-op variable
• When to change protocol?
  – \#failed test-and-test&set attempts > threshold \( \rightarrow \) TTS\( \rightarrow \)QUEUE
  – \#successive fetch-op attempts when \{waiting time on queue > time limit\} > threshold \( \rightarrow \) QUEUE\( \rightarrow \)combining tree
  – #fetch-and-op requests reaching root without combining with enough fetch-and-op requests > threshold \( \rightarrow \) combining tree\( \rightarrow \)QUEUE
  – #successive fetch-and-op attempts when queue empty > threshold \( \rightarrow \) QUEUE\( \rightarrow \)TTS
Reactive Message-Passing

- Shared memory vs. message-passing
  - Shared memory on top of message-passing
  - test-and-test&set vs. message-passing queue locks
  - test-and-test&set lock vs. centralized message-passing vs. software combining tree fetch-and-op

Experimental Setup

- Alewife machine
  - Message passing
  - 2D mesh network
  - LimitLESS cache coherence protocol
  - Atomic fetch-and-store hardware primitive
- Cycle accurate simulator
- 16-node Alewife machine
- Applications
  - Gamteb (9 counters using fetch-and-increment)
  - Traveling Salesman Problem (fetch-and-increment for access to global task queue)
  - Adaptive Quadrature (similar task queue to TSP, but lower contention)
  - MP3D (a lock used for atomic update of collision counts, other lock for atomic update of cell parameters)
  - Cholesky (not important)

Results: Baseline Performance

**Spin Lock**
1. Acquire lock
2. Execute 100 cycle critical section
3. Release lock
4. Delay for 0-500 cycles
5. Repeat 1-4

**Fetch-and-Op**
1. Fetch-and-increment
2. Delay for 0-500 cycles
3. Repeat 1-2
(Radix-2 combining tree with 64 leaves)

Results: Applications (Fetch-and-Op)

- Choice of fetch-and-op algorithm important
- Gamteb: Reactive algorithm best for 128 processors due to different protocols for different counters
Results: Applications (Spin Locks)

- Queue lock good for all contention levels
  - Coarse grained computation
- MP3D
  - Queue lock for collision counts update

Results: Reactive Message-Passing (Baseline)

Results: Reactive Message-Passing

- 16 processors
- Normalized w.r.t. queue lock
- Contention levels varying frequently ➔ poor performance
  - Overhead dominates

Discussion