Understanding Tradeoffs in Software Transactional Memory

Dave Dice

Sun Microsystems JVM Core Engineering

Nir Shavit

Tel-Aviv University

Sun Labs Scalable Synchronization Research Group

Concurrency Today

- We wanted more clock speed ...
  - Instead we got more cores
  - Moore restated: cores instead of transistors
- Niagara-2 – 64x
- Thread-level explicit parallelism …
  - **not** a feature
  - it's a remedy with side effects - complexity
  - Best remaining avenue
  - Patterson: end of La-z-boy era

Harnessing Concurrency

- Locks
  - Deadlock & composability
  - Broken variable::lock mappings
  - Fine-grained → fast & complex
    - Error-prone – best left to experts
  - Coarse-grained → slow & simple & safe
    - Typically untapped parallelism
- Non-blocking : wait-free and lock-free techniques
  - Complex – not suitable for most programmers
  - Performance varies - progress
  - Small catalog of known-good algorithms
    - concurrent collections

Human Scalability

- Programmers - Programs
- Reduce complexity
- Eliminate sources of error
- Raise abstraction level above locks
- Think sequentially, execute concurrently
- The right constructs to use concurrency
- Still provide scalability & performance

Transactional Memory

- Synchronization mechanism
- Library-based until recently
- Should be integrated into language
- Often expressed as “**atomic {…}**”
- Varieties: Hardware, Software (STM), hybrid
- STM design issues impact
  - Compiler
  - Runtime environment
- **Pluggable** STM implementations

Software Transactional Memory

- Optimistic concurrency control
  - Detect and recover from conflicts
- Speculative phase – run transaction
  - Track loads – read-set
  - Save stores – write-set
- Followed by commit attempt – atomic
  - Validate the read-set
    - Check for concurrent interference
  - Commit the speculative stores
  - Otherwise abort & retry

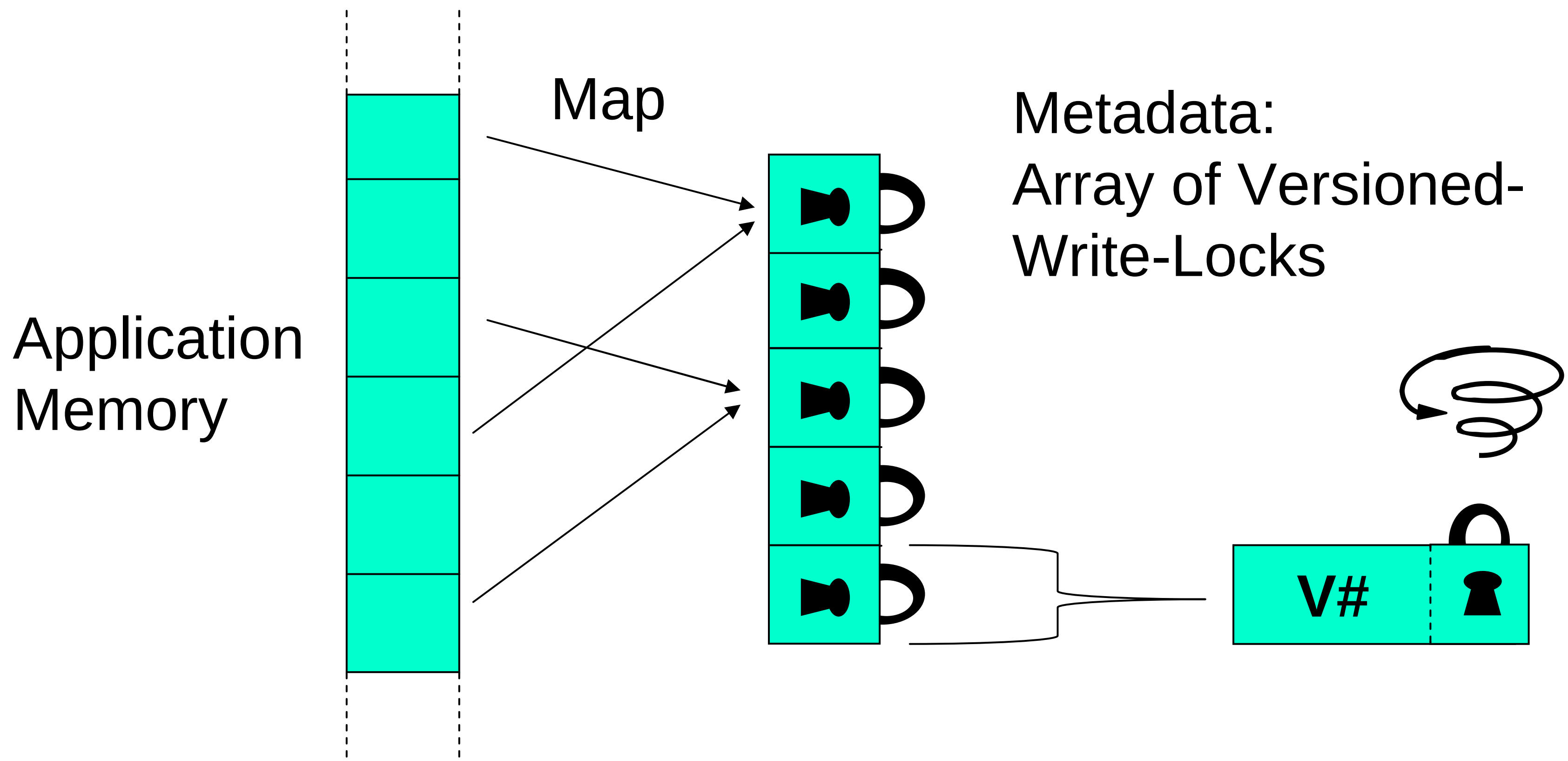
Lock-Based STMs

- lock-free v. **lock-based** implementation
- Optimistic concurrency implemented with locks
- Advocated by Ennals
- Solaris schedctl makes it viable
  - Advisory preemption deferral
- Transactional Locking: TL and TL2 …

Lock-based STM Design Choices

- Lock :: variable mapping
  - **Separate array of locks**
    - Array size, hash & stripe-width
  - Colocate lock with data : per-object
- When to acquire locks
  - As encountered or at **commit**-time
  - Scalability – reduced lock-hold times
- Store policy
  - Update-in-place vs **speculative store buffer**
- Read-set consistency during a transaction
  - Prevent inconsistent execution
  - Allow but detect & recover

TL Data Structures



Shared variable is <u>covered</u> by a single lock
Hash function: maps variable ➔ lock

TL

- Data structures:
  - Thread-local read-set and write-set
  - Array of locks
- **Store**
  - Save (address,value) in write-set
- **Load**
  - Look-aside in write-buffer
    - RAW Hazard
    - Accelerate with Bloom filter
  - Load both lock and variable
  - Check lock-bit
  - Record (address,version#) in read-set

TL

- **Commit**
  - Acquire locks covering write-set
    - schedctl
  - Bounded spin, then abort, back-off, retry
  - Validate read-set version#s unchanged
  - Write-back write-set
  - Increment and release write-set locks
    - Locks held for a very short time
- **Periodically validate** read-set
  - During speculative phase
  - Seen inconsistent read-set? Abort

Inconsistent Execution

↓ Time

Tx1	Tx2
{ X=5/(A+B) }	{ A=4;B=0; }
TLD(A) 0	TST(A=4)
	TST(B=0)
	Commit
	Lock A,B
	Validate
	ST A=4;B=0
	Unlock A,B
TLD(B) 0	Success
5/(A+B) :Trap!	

- Initially: A=0;B=1
- Invariant: (A+B)!=0
- A,B shared txl
- X local variable
- Tx1,Tx2 interleave

Zombie Transactions

- Has seen an inconsistent read-set
- Fated to abort
- But still running app code
- Misbehavior:
  - infinite loops : compiler must emit checks
  - Traps : runtime must tolerate
- Unsafe in an unmanaged runtime environment

Zombies - Alternatives

- Validate periodically: admits zombies
- Validate after each transactional load
  - Prevents zombies
  - Cost is quadratic with read-set size
- Read-write locks - form of visible readers
  - Acquire read-lock before load
  - Read-set always consistent
  - No zombies thus no validation required
  - Atomics and coherency traffic (writes)
  - Admits less parallelism - scalability
- **TL2** – prevents zombie execution

TL2

- Successor to TL - DISC'06 [+Ori Shalev]
- Efficient validation
  - No zombies
  - Avoids quadratic cost
  - Avoids visible readers
- Less intrusive to code-generation and runtime environment
- Key: global clock
  - Hardware or software
  - Thread-local wv, rv variables and global clock

TL2 - Algorithm

- **Start**: rv = globalclock
- **Load**: Same except …
  - check version#(variable) <= rv
- **Store**: same as TL
- **Commit**:
  - Acquire locks on write-set
  - Validate read-set version#s <= rv
  - wv = Fetch&Add (globalclock)
  - Write-back
  - Store wv into locks covering write-set
    - Releases locks and updates version#

Memory Lifecycle Pathology

Time ↓

Tx1	Tx2
{ if(A!=null) A->Field=3}	{ T=A;A=null;}
TLD(A) non-null	TLD(A); T=A;
TST(A->Field=3)	TST(A=null)
Commit	
Lock A->Field	
Validate A	Commit
	Lock A
	Validate A
	ST A=null
	Success
	free(T)
ST A->Field=3 (!)	

Lifecycle Concerns

- Hazard:
  - Memory region is accessed transactionally
  - Region removed from transactional data structure
  - Then accessed non-transactionally
  - Latent transactional stores
- Explicit privatization
  - TL & TL2 : <u>quiesce</u> regions
  - Wait for latent stores to complete
- Implicit privatization
  - Possibly less scalable (today)
  - Easier for programmer – reduced complexity

Compiler Integration

- Hybrid Transactional Memory
  - ASPLOS 2006 [Damron, et al.]
- Prototyped in production C++ compiler
- No changes to data layout
- No GC required
- Now supports TL2
- Pluggable STMs: HyTM, TL2

Summary

- STM design decisions impact code generation
  - Runtime & JIT coevolved with GC – now TM
- TL or TL2: managed runtimes -- Java
- TL2 : unmanaged environments -- C/C++
- Competitive with hand-coded performance
- Lifecycle issues
- Schedctl makes blocking STMs viable

Thank You