Eraser: A Dynamic Data Race Detector for Multithreaded Programs

STEFAN SAVAGE, MICHAEL BURROWS, GREG NELSON, PATRICK SOBALVARRO, and THOMAS ANDERSON

Ethan Holder 2014

Presentation Outline

- About the Authors
- What is Eraser?
- Why is Eraser Important?
- Background & Related Work
  - Happens-Before
- Eraser Algorithm
- Eraser Implementation
- Experiences
- Evaluation

VirginiaTech
Invent the Future

About the Authors

- Savage and Anderson (UCB) were at U. Washington. Burrows, Nelson, and Sobalvarro were at Digital Equipment Corporation.
- Cited by 387 (according to ACM DL) including RaceMob and PARROT.
- Each averages over 35 cites per article.
- Eraser their most or second most cited article.

What is Eraser?

- Detects data races in a multithreaded program dynamically.
- Allows developers to easily find and correct concurrency bugs.
- Works by monitoring data accesses at a very low level and observing locking patterns.

Why is Eraser Important?

- Basically the first to use lock set idea instead of Happens-Before.
- Lock set is relatively low overhead compared to Happens-Before, albeit less accurate.
- Leads to future work combining the two approaches for both speed and reliability (FastTrack by Flanagan and Freund in PLDI 2009).

Background

- What's a data race?
- One (or more) thread(s) writes as another thread tries to read or write at the same time.
- No synchronization mechanism (lock).
- Unclear who executes first and what final value is read/written.
- Very hard to debug!

Related Work

- Largely compared to Happens-Before relationships by Lamport in 1978.
- Monitors by Hoare in 1974.
- Lock covers by Dinning and Schonberg in 1991.
- NOT Eraser by Mellon and Crummey in 1993.

Happens-Before

- Within a thread, execution order generates event order.
- Between threads, synchronization events generate a partial order based on shared resource access.

Happens-Before

	Thread X	Thread Y	Thread Z
Time 0	Lock(a) Modify Unlock(a)		Lock(b) Modify Unlock(b)
		Lock(a) Modify Unlock(a)	
…			Lock(b) Modify Unlock(b)
	Lock(a) Modify Unlock(a)		
Time n		Lock(a) Modify Unlock(a)	

Eraser Algorithm

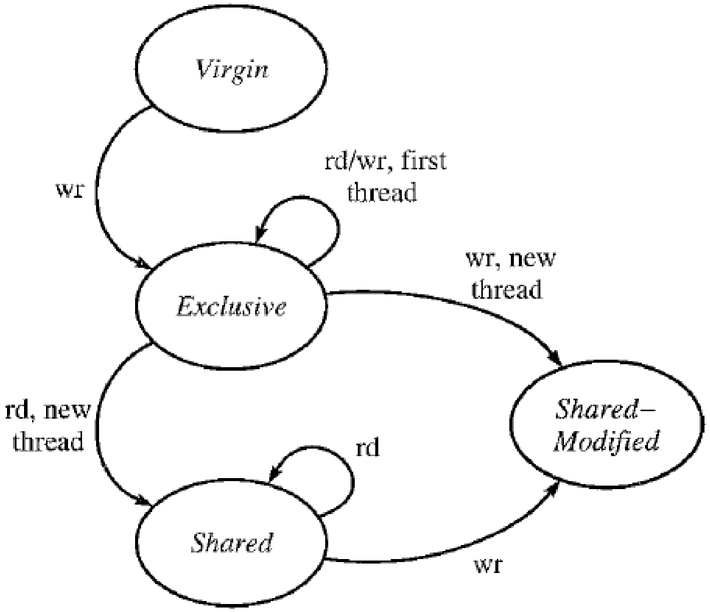
- Initialize set of candidate locks to all locks for each shared variable and held locks to empty set.
- On an access, change the set of candidate locks to be the former set's intersection with the currently held locks.

Example

	Thread X	Thread Y	Held Locks	Candidates
Time 0			{}	U
	Lock(a) Modify(var1) Unlock(a)		a	U^a = a
…				
		Lock(b) Modify(var1) Unlock(b)	b	a^b = {}
Time n				

Improvement

- Ignore initialization.
- Allow read sharing.
- Only remove read locks on the second thread write.



Failure

	Thread X	Thread Y	Held Locks | Candidates
Time 0	Allocate(var1) Set thread Y to modify var1 Start thread Y		{} | U
…	Modify(var1)	Modify(var1)	
	If this mod is 1st, no race detected.	If this mod is 1st, race detected and output.	
Time n			

Potential for race regardless.

VirginiaTech Invent the Future

Eraser Implementation

- Instruments loads and stores for checking the set of candidate locks.
- Instruments lock acquire and release and thread initialization and finalization for checking locks held.
- Instruments storage allocator calls for dynamic data.

Output

- On a race detection (no locks held during access) the following is reported:
  - File and line number.
  - Backtrace of stack frames.
  - Thread ID
  - Memory Address and Access Type
  - Program Counter and Stack Pointer

Shadow Words

- Correspond to real data words in stack and heap.
- Contain lockset index (30-bit) or thread ID and state condition (2-bit).
- Lockset index points to hash table entry for the distinct set of locks held.

VirginiaTech
Invent the Future

False Alarm Annotations

- EraserIgnoreOn/Off for disabling benign race output.
- EraserReuse for resetting shadow memory when using internal memory allocators.
- EraserReadLock/Unlock and EraserWriteLock/Unlock for communicating private lock usage.

AltaVista Experience

- Programs Tested
  - Ni2 (9) and ft (5)
  - Mhttpd (10)
- False Positives
  - Avoid locking overhead
  - Finalization checks
  - Global Statistics

Vesta Cache Server Experience

- Valid Fingerprint Boolean
- False Positives (10)
  - CacheS Free List Flush
  - TCP_sock and SRPC Objects

Petal Experience

- Server Running State Checks
- False Positives (several)
  - Private Reader-Writer Locks
  - Global Statistics
  - Stack Memory Reuse on Forking

Undergraduate Assignments Experience

- 10% data races detected across runnable assignments.
- False Positives (1)
  - Locked Head and Tail in Queue

Experiences

- Dependent on test runs versus absolute relationships.
- “Easy to use”
- Promising results with deadlock-checking.

Evaluation

- Novelty
  - Lock sets instead of Happens-Before.
- Importance
  - Much faster.
  - Handles dynamic data.
  - Works on large and small scale code bases.

Evaluation

- Negatives
  - Still not that fast.
  - Doesn’t specify how to fix the problem (later papers actually tell how or do it themselves).
  - Measurements aren’t similar and are arbitrarily ran.
  - LOTS of false positives.

Questions

?

Virginia Tech
Invent the Future