RaceTrack: Efficient Detection of Data Race Conditions via Adaptive Tracking

If the lock-free code happens to be correct, these warnings are false alarms. However, it is difficult even for experienced programmers to write correct lock-free code.

Implementing RaceTrack entirely inside the CLR, although an admittedly rather technical undertaking for a very sophisticated runtime, makes it completely transparent to applications.
Yuan Yu, Tom Rodeheffer, Wei Chen

1 Overview of data race detectors

A data race occurs in a multithreaded program when two threads access the same memory location without any intervening synchronization operations, and at least one of the accesses is a write.

- **Static analysis** tends to be conservative, generating spurious warnings (false positives). Scaling is difficult because it is a whole program analysis. Strong type systems can be used to avoid data races but they require source annotations and restrict concurrency models.
- **Dynamic analysis** has fewer false positives, but the analysis is limited to the thread interleavings that are executed.
- **Lockset algorithms** ensure that a piece of data is accessed with a consistent set of locks held. Does not recognize fork/join parallelism and asynchronous calls. With no language support, requires shadow words (high overhead).
- **Happens-before** data race detectors use program order and synchronization events to establish a partial temporal order on program statements. If two memory accesses are not temporally ordered, they can race.
- **Language independent** program instrumentation intercepts every load and store to global memory, and often must have a shadow word for every word of global memory to hold lockset information. 10–30× slowdown.
- **Object-oriented** detectors can detect races at object granularity, lowering overheads (2× in time), while increasing the probability of false positives.

2 RaceTrack

- Implemented in Microsoft's Common Language Runtime.
- Modifies JIT and garbage collector.
- Dynamically adjust object to field granularity. Deals with arrays.
- Dynamically adjusts tracking lockset and happens-before relationship.
- 3× slowdown, with 1.2× memory use on SpecJBB.
- Deals with interactions between managed and native code. This is an industrial strength project!

3 Vector clocks

- Lamport clock defines the happens-before relation. If u causally happens before v ($u < v$) then the Lamport clock value of u will be less than the Lamport clock value for v. But if $L(x) < L(y)$ we **cannot** conclude that $x < y$.
- With a vector clock, each node includes the last known timestamp from every other node. With vector clocks if $V(x) < V(y)$ then $x < y$. Events are simultaneous ($u||v$) iff $V(u) \not< V(v) V(v) \not< V(u)$. Note that the simultaneity relation $||$ is not transitive. $V(x) < V(y)$ iff $V(x)[i] < V(y)[i]$ where i indexes the vector.

4 Detecting races

- See notes for definition of data race and lockset.
- Each *thread* has a lockset (L_t)and a vector clock (B_t).
- Each *variable* has a lockset (C_x) and threadset (S_x). C_x initialized to all possible locks, S_x initialized to $\emptyset$.
- Only the most recent access for a given thread is kept.
- Vector clocks identify accesses that are concurrent.
- Only allow lockset to grow when threadset has multiple entries.
- See figure 2 for basic algorithm.
- Only issue a warning when $|S_x| > 1$ and C_x is empty.
- See figure 3 for an example where the lockset algorithm would

5 Making detection practical

- Use state machines to limit the amount of tracked data for each variable.
- See Figure 4 for state machine.
- Exclusive2 is the state where lockset reports race.
- Adaptive granularity (on potential race, track fields/array elements).
- Funny idioms
  - Master thread checks IsAlive to find dead threads. This must be treated as a thread join.
  - Finalizers are reset to state Exclusive0
- Implementation
  - Immobile lockset table. Realloc on filling table (the number of lockset don't increase without bound).
  - Immobile threadset table. Reference counting to garbage collect.
  - Pointer to racetrack state at the end of the object to preserve layout.
  - Figure 6 shows the space-optmized state layout, with lock-free updates. Sometimes, only the first word needs updating.
- Don't track volatiles.
- Track first, middle and last m elements of an array.
- Discard some elements of a large vector clock.

For Boxwood and SATsolver slowdown was based on elapsed runtime, for SpecJBB on its reported throughput, for Crawler on the number of pages downloaded, and for Vclient on average CPU load.

6 Making racetack usable

- Generate a stack trace of first racing thread on second race occurrence.
- Generate lots of warnings, then prioritize using heuristics. Allow user to supress warnings.
- See Table 1 for taxonomy of races.
- See results examples of real races.
- See Table 2 for overheads.