Write Barrier Elision for Concurrent Garbage Collectors

Martin T. Vechev
Cambridge University

David F. Bacon
IBM T.J. Watson Research Center
Write Barriers for Concurrent GC

- Mutator and Collector interleaved
- Mutator changes object graph
- Potential race conditions
- Write barriers a form of synchronization
  - Between mutators and the collector.
Can Synchronization Be Reduced?

- Minimum information from a mutator to a collector?
  - without compromising correctness?
- Can static analysis help?
- Can we discover dynamic opportunities?
Outline

• The Synchronization Problem
• Elimination conditions
• Elimination opportunities (limit study)
• Elimination correlation
• Exploiting Order
• Conclusions and Future Work
The Synchronization Problem

Collector Working
- Marks B as live
The Synchronization Problem

- Installs P2
- Marks B as live

Collector Working
Thread Working

- Installs P2
The Synchronization Problem

- Installs P2
- Deletes P1
- Marks B as live
- Installs P2

Collector Working
- Marks B as live

Thread Working
- Installs P2
- Deletes P1
The Synchronization Problem

- Installs P2
- Deletes P1
- Marks B as live
- C was not seen
- Reclaims C: \textbf{live!}
The Synchronization Problem

A and B contain pointers to Object C

Point of Error
Types of Write Barriers

Steele

Marks B (Source) for rescanning

Dijkstra

Mark C (new target) as live

Yuasa

Mark C (old target) as live

Collector Working

Thread Working

Collector Working

Thread Working

Collector Working
Costs of Concurrent Write Barriers

• Mutator Overhead
  – Direct Run-time Overhead (5-20%)
  – I-cache pollution by write barrier code

• Collector Overhead
  – Process Write Barrier Information

• Space costs
  – Sequential Store Buffer
  – Increased Code Size

• Termination Issues
  – Yuasa vs. Dijkstra/Steele
Contributions

• **Four Elimination Conditions** a static analysis can utilize.
  – Main idea based on pointer lifetimes.

 1. **Two Covering conditions**: Apply to all barriers

 2. **Two Allocation conditions**: Apply to incremental barriers.

• **Limit study** shows potential for barrier elimination
  – For Yuasa, on average 83% can be eliminated
  – For Dijkstra and Steele, on average 54% can be eliminated

• **Correlation** study between barrier elimination and
  – Object Size
  – Object Lifetime
  – Program Locality
Key Observation:

- P1’s lifetime covers P2’s
- Barriers on P2 are redundant
Single Covering Condition (Incremental)
Single Covering Condition (Snapshot)
Key Observation:

- P1 together with a barrier on P2 form a virtual pointer PV
- Apply Single Covering to eliminate barriers on P3
Multiple Covering Condition (Incremental)
Multiple Covering Condition (Incremental)
Multiple Covering Condition (Snapshot)
Multiple Covering Condition (Snapshot)
Single Allocation Condition

- Allocation conditions exploit Initial Root Set Marking
- Main idea: At the time of a barrier, the object is already marked
- Only if allocating non-white (also trade off), for Dijkstra/Steele only

Key Observation:
H starts Inside ‘New’ pointer N
Multiple Allocation Condition

Key Observation:

- $V = N \text{ join } L$
- Apply SAC between $V$ and $H$
Eliminating Null Pointer Stores

Yuasa(destination, old_pointer)
if ( Collector_Marking
    && old_pointer != NULL)
store (old_pointer);

Dijkstra(destination, new_pointer)
if (Collector_Marking
    && new_pointer != NULL)
store (new_pointer);

- If old pointer is NULL, eliminate barrier
- Yuasa vs Steele/Dijkstra
- Yuasa good because of initialization
- Null stores are easier to eliminate
  - But less payoff
Evaluation Methodology

• Limit study based on elimination condition

• Shows potential for elimination

• Traces
  – Jikes RVM 2.2.0
  – Trace Events: Allocation, Pointer Stores
  – Application Objects Only

• Which benchmarks:
  – SpecJVM98 (-s100)
  – Jolden
  – Ipsixql
  – Xalan
  – Deltablu
Barrier Elimination Potential – Dijkstra / Steele

Geo Mean

power
deltablue
bh
health
ipsixql
xalan
compress
db
javac
mtrt
jack
Barrier Elimination Potential - Yuasa
Time (MB) vs. Eliminated Yuasa Barriers

- Elimination is Periodic => WB elimination occurs in bursts
Time (MB) vs. Eliminated Yuasa Barriers

- Elimination is Periodic $\Rightarrow$ WB elimination occurs in bursts
Object Size (words) vs. Eliminated Yuasa Barriers

- Elimination is on Small objects
Object Age vs. Eliminated Yuasa Barriers

• Elimination is mostly on Young objects (Log Y scale)
Object Age vs. Eliminated Yuasa Barriers

• An Exception
Further Write Barrier Elimination

• So far assumed collector sees pointers in any order

• Often, Collector order exists
  – take advantage of it

• Main idea: writes on the collector wave are safe.
  – Cannot apply previous conditions
  – Can eliminate more barriers

• Order extends to Lists, Queues, Trees (Needs connectivity approximation)
Single Object Scenarios (Order matters)

Broken Sequence

Collector Working
- B is partially scanned (gray)

Thread Working
- Installs P2

Thread Working
- Deletes P1

Collector Working
- Reclaims live object C incorrectly
Correct Sequence

- Collector Working
- B is partially scanned (gray)
- Thread Working
- Installs P2
- Thread Working
- Deletes P1
- Collector Working
- C is NOT collected
Conclusions and Future Work

• Static Analysis Elimination Conditions

• An Upper Bound

• Hot Barrier Methods

• Yuasa Barriers seem superior to Dijkstra/Steele
  – (study higher elimination vs. reduced floating garbage)
  – Yuasa are harder to statically eliminate

• More conditions possible
  – Requires formal reasoning

• Other elimination combinations possible:
  – Coverage (ownership) and Order can be exploited further.