An efficient, incremental, automatic garbage collector

Deutsch & Bobrow

Presented by Hou-Jen Ko

Outline

- Introduction
- Advantage/Disadvantage of Reference counting
- Methodology
- Conclusion

Introduction – Reference counting

- Reference counting (RC) contains
  - A counter for each object
- The counter means how many other objects reference it.
- When the value of the counter becomes 0
  - The object is garbage and can be reclaimed.

Advantage of RC

- GC costs are distributed throughout the computation
- Unreferenced structures can be reclaimed immediately
- Do not need to know the roots of the program
    - It can reclaim some memory even if some parts of the system are unavailable

Disadvantage of RC

- Redefine all read / write operations in order to manipulate reference counts
  - Undesirable, i.e. traverse a list
- Atomic operations to update reference counts
- Circular structures are not reclaimed
- Space overhead to store reference counts

Three ways to improve RC

- ***Deferral*** *RC*
  - Defer the identification of garbage objects to a reclamation phase
- ***Coalescing***
  - Eliminate unnecessary temporal adjustments of reference counts
- ***Buffering***
  - Buffer the adjustment of reference counts for later processing

Deferred RC

- Observation:
  - The majority of objects have a RC of 1
- Solution:
  - Record transactions that affect the accessibility in the *transaction file*
    - ***Allocate*** a new cell
    - ***Create*** a pointer to a cell
    - **Destroy** a pointer to a cell
  - Store the reference count for objects with 2 or more references in a multireference table (MRT)

Deferred RC

- Observation:
  - The majority of pointer loads are to local and temporary variables
- Solution:
  - Do not count local variable references
    - Counts are no longer accurate
  - Need a *zero count table* (ZCT) for objects with reference counts of 0 but may be referenced from locals

Transaction file processing

```
for ( Transaction* t = FirstTransaction(); t; t = Next(t) ) {
    if ( IsAllocate(t) ) {
        AddToZCT(t);                        // Upon allocation, simply add to ZCT
    } else if ( IsCreatePtr(t) ) {
        if ( !RemoveFromZCT(t) ) {          // Remove from ZCT if present
            if ( IsInMRT(t) )
                IncrementMRTReference(t);   // Add reference to MRT or increment its ref count
            else
                SetMRTReferenceCount(t,2);
        }
    } else if ( IsDestroyPtr(t) ) {
        if ( IsInMRT(t) ) {
            if ( GetMRTCount(t) == 2 )
                RemoveFromMRT(t);
            else
                DecrementMRTReference(t);
        } else {
            AddToZCT(t);                    // Add to ZCT if this was the last reference
        }
    }
}
```

Reclamation

- Reference count = 0 (in ZCT)
  - Not count yet referenced from local variables
  - “**May**” be live
- Any objects in ZCT are reclaimable if it is not reachable from the stack.
  - Variable reference table (VRT) contains all the pointers from the stack.
  - Reclaim (obj ∈ ZCT) **if** IsInVRT(obj) = true

Reclamation

- Upon freeing an object
  - Decrement the references counts of objects which it may point to
- Free a large data structure
  - May defer the decremented operations in order to disperse the disruptive effects

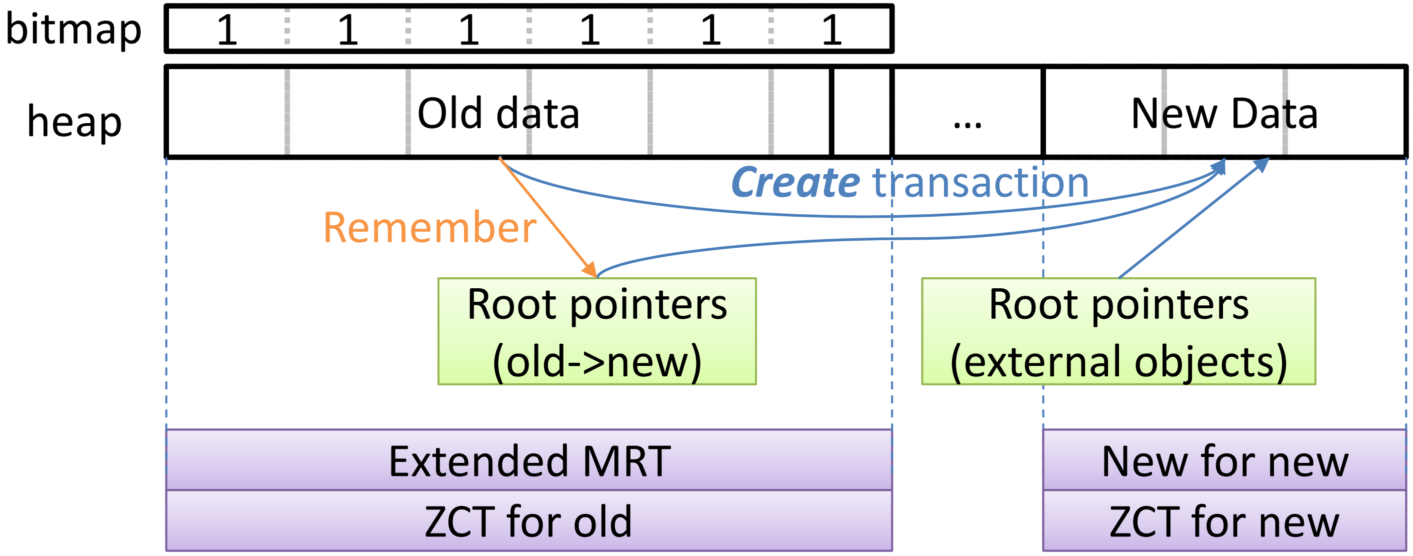
Coalescing

- Eliminate unnecessary temporal adjustments of reference counts
  - – allocate – create
  - – destroy- create

Linearizing Garbage Collection

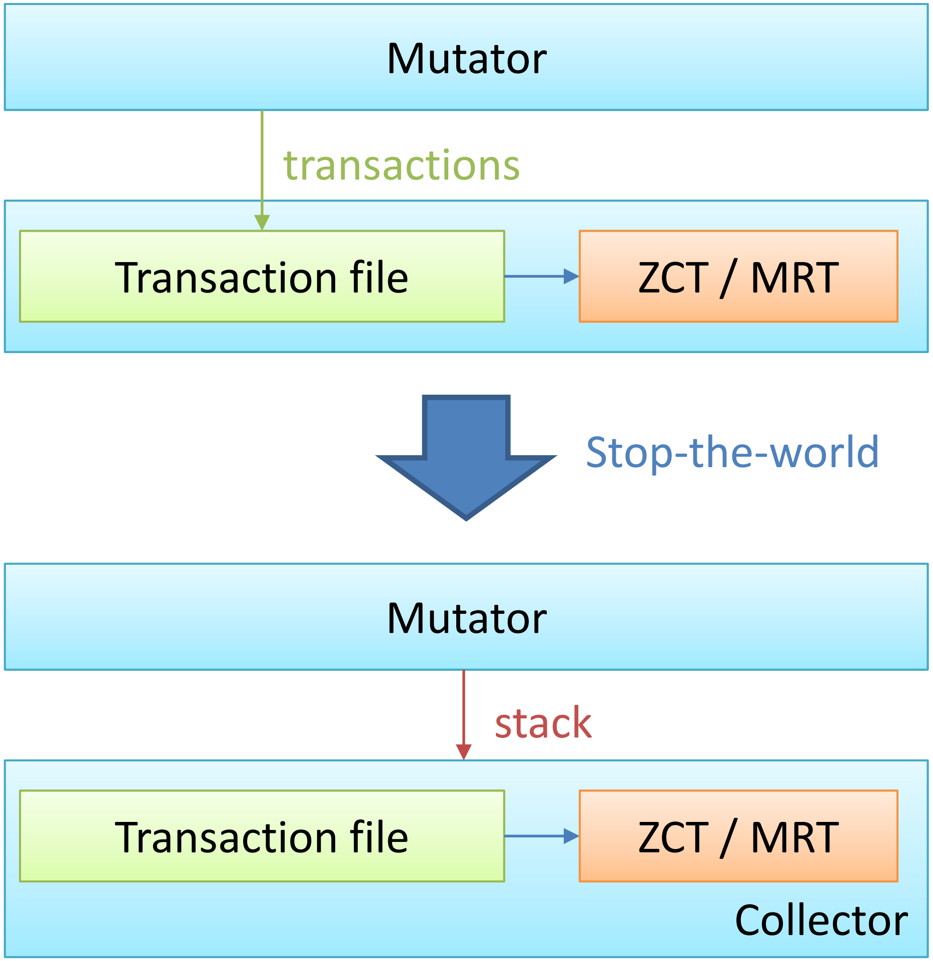
- Collect circular structures
- Generational garbage collector
  - In depth-first traversal order
  - Extended MRT
    - MRT must be updated before copying
    - (taking the stack into account)
    - Store the relocation address for each entry of MRT (?)
- It improves locality

Incremental linearization



- Old data can be released by reference counts
- Full linearization will be performed if the storage needs to be reclaimed
- MRT and ZCT are segmented by address

Further Observation



Reference

- www.cs.utexas.edu/users/mckinley/395Tmm/talks/Mar-9-DRC.pdf
- Richard Jones, Antony Hosking, and Eliot Moss, *The Garbage Collection Handbook*