A Description of the RCImmix Algorithm

Reference Counting with better heap allocation

Nathan Jervis

Department of Computer Science
McMaster University, Hamilton
jervisnd@mcmaster.ca
1211159

March 26, 2014

Overview

- Introduction to automatic memory management
- Problems with existing reference counting
- Optimizations to reference counting
- RCImmix algorithm

Manual Memory Management

Manual Memory Management

- Difficult to use
- Can cause dangling pointers
- Leads to memory leaks

Much better if the compiler/runtime can manage memory for us

Automatic Memory Management

Tracing Garbage Collector:

- Periodically pause program and follow program references
- Collect anything not referred to

Reference Counting:

- Counter keeps track of how many things are pointing to it
- When counter reaches 0, free memory
- `Allocate`, `Retain`, `Release`

Reference Counting vs Tracing

Tracing Garbage Collector:

- Little work for allocation
- Better cache performance (with compacting)
- Requires pausing to collect

Reference Counting:

- Doesn't pause
- Huge overhead
- Poor cache locality

RCImmix Optimizations

Optimizations

- Tracing collector as a backup
- Limited Bit Count
- Block Based allocation
- Nursery Allocation and Copying Collectors
- Objects are born as dead

Tracing Backup Collector

- Naive reference counting won’t collect cycles
- Reference counting handle normal memory collection
- When cycle garbage accumulates too much, call Tracer
- Tracer will very rarely be called, so you still don’t have to worry about pauses too much

Limited bit count

- In theory everything could point to object, requires 32/64 bits
- In practice most objects only a few things pointing to it
- Using 3-4 bits is fine
- If it overflows, just leave it at max, don't decrement
- Tracer will fix it when it collects

Block Based Allocation

- Items are allocated out of a block
- Block keeps a pointer to the next free spot
- Count of number of live objects on the block
- Great for cache performance

Nurseries and Copying Collection

- Objects are allocated into a nursery
- Move when mature (copied or pass collection cycle)
- Only new objects in nursery
    - Will be collected during collection cycles
    - Most objects die young, not copied
    - Nurseries are cheap way to collect

Objects are born as dead

- When you create the object, garbage collector already considers it dead
- Only when it moves or matures do you consider it alive
- ModBuffer contains all objects that have created new objects
- Can process ModBuffer to check if ”dead” objects are actually alive

Typical Use

Algorithm 1.1: User Code(*args*)

```
main
  $x \leftarrow$ Allocate($size$)
  . . .
  Retain($x$)
  $y \leftarrow x$
  . . .
  Release($y$)
  . . .
  release($x$)
```

Allocate

Algorithm 2.2: ALLOCATE(*size*)

```
global $bumpPointer, block$

if $size < block.end - bumpPointer$
   then $\begin{cases} pointer \leftarrow bumpPointer \\ bumpPointer \leftarrow bumpPointer + size + 1 \end{cases}$

   else $\begin{cases} block \leftarrow \text{GetNewFreeBlock}() \\ \textbf{return } (\text{Allocate}(size)) \end{cases}$

$pointer.new \leftarrow$ true
$pointer.count \leftarrow 0$
return $(pointer)$
```

Retain

Algorithm 2.3: RETAIN(*object*)

```
if $object.new =$ false
  then if $object.count$ not $max$
         then $object.count \leftarrow object.count + 1$
       return
$pointer \leftarrow$ COPYTONEWLOCATION($object$)
ADDTOMODBUFFER($object$)
$object.count \leftarrow 2$
$object.block.liveCount \leftarrow object.block.liveCount + 1$
return
```

Release

Algorithm 2.4: RELEASE(*object*)

```
if $object.new = $ true
  then return
if $object.count$ not $max$
  then $object.count \leftarrow object.count - 1$
if $object.count = 0$
  then { PROCESSMODBUFFER()
         $object.block.liveCount \leftarrow object.block.liveCount - 1$
         FREEBLOCKS()
```

ModBuffer

Algorithm 3.5: ADDTOMODBUFFER(*object*)

```
global ModBuffer
ModBuffer.Push(object)
```

Algorithm 3.6: PROCESSMODBUFFER(*none*)

```
global ModBuffer
for each $obj \in ModBuffer$
    do {for each $child \in obj.references$
          do {if $child.new =$ true
                then {Retain(child)$child.count \leftarrow 1$
```

Conclusion

	Reference Counting	Tracing	RCImmix (Both)
Overhead	Significant	None	Minimal
Speed	Slow	Fast	Fast
Cache Performance	Poor	Good	Good
Pauses	Short	Long	Mostly short
Implementation Difficulty	Easy	Hard	Very Hard

Resources

- Xi Yang Kathryn S. McKinley Rifat Shahriyar, Stephen M. Blackburn.
  Taking off the gloves with reference counting immix.
  pages 1–18, 2013.

- Daniel Frampton Rifat Shahriyar, Stephen M. Blackburn.
  Down for the count? getting reference counting back in the ring.
  pages 1–11, 2012.