Extending Balanced Garbage Collection by Reference Counting

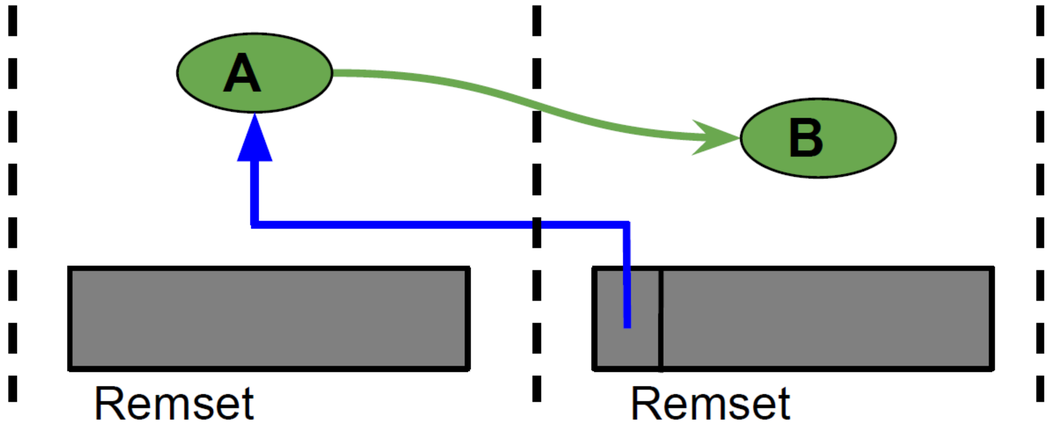
Johannes Ilisei, Kenneth B. Kent, Gerhard W. Dueck
University of New Brunswick, Faculty of Computer Science
jilisei@unb.ca, ken@unb.ca, gdueck@unb.ca

Outline

Optimizing throughput and improving the execution time of garbage collection (GC) is nowadays one of the main goals in designing efficient and competitive Java Virtual Machines. IBM's balanced GC introduces a region-based heap layout with a copying collector and aims to avoid whole heap collections. This project proposes to extend balanced GC by a reference counting (RC) algorithm in order to decrease GC time and increase the amount of freed memory.

Problem Description

One problem with balanced GC is the occasional copying of technically dead objects that seem alive in the eye of the collector. This is due to references from unreachable objects to children that are located in different regions. Considering this simple example with two objects:



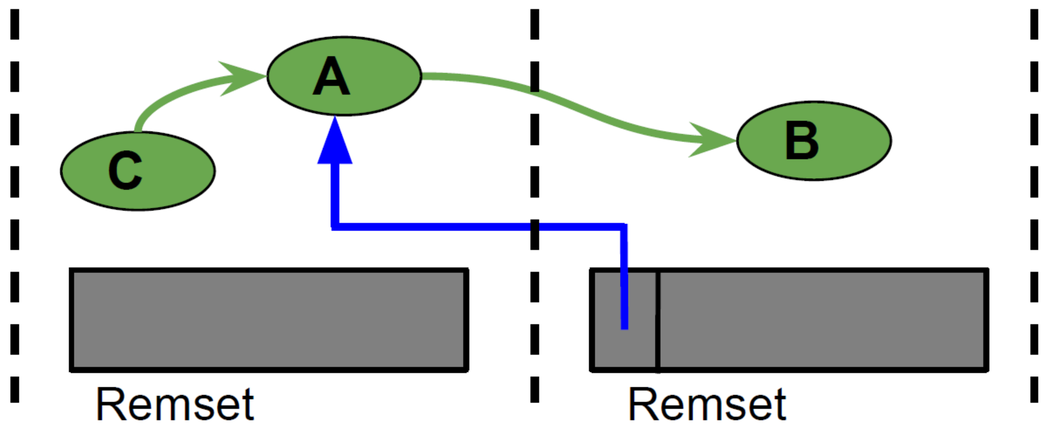
- A and B are Objects from different regions.
- A is pointing to B therefore the remset from the second region contains a pointer to A.
- A does not have any incoming references so it is dead.
- B is technically dead as well (unreachable from any live object).
- If the second region gets collected during partial GC, the remset entry to Object A will lead to the copying of B.
- B can only be freed if A gets freed first or both regions are collected simultaneously during a partial GC or a global heap collection.

Project Proposal

An easy solution to the problem of copying dead objects is achieved by adding RC to the balanced GC policy. Considering the first example again:

- The reference count of A equals zero.
- During collection, before following the children from parents, determine the reference count of the parent object (A) and only continue if the reference count is greater than zero.
- Thereby object B will not be copied.

This solution does not solve the problem in every case. If we consider this next scenario:



- Now A's reference count equals one although technically it is still dead.
- Three possible solutions on how to collect B:
  1) Just copy B to a new region.
  2) Implement a smarter reference count algorithm that discovers during mutation that the reference from C to A is irrelevant.
  3) Freeing unreachable objects like C during mutation.

Further Possibilities and Advantages through RC

- Select regions for collection based on number of incoming references by counting references per region.
  - Free regions with few incoming references.
- Determine the level of fragmentation by comparing incoming references to used memory per region.
- Group objects together by number of references instead by age.

Implementation and Testing

The RC algorithm will be added to balanced GC in a JVM simulator programmed in C++. Execution times, throughput, freed memory, and other characteristics will be compared to balanced GC and other collectors in order to determine the beneficial effect of this new approach.