CSC 548: Lecture 10

Overview

Copying gc

Implementing gc

Generational gc

Incremental gc

GC and the memory hierarchy

Implementing gc

For each object on the heap, we need to know which fields are pointers, and which fields are base types. Why? How can we do this?

For each frame on the stack, we need to know which fields are pointers. Why? How?

For copying gc, where can we put the forwarding pointer?

Note that this requires some help from the compiler. What can we do if the compiler won't play ball? (For example, can we gc C programs?)

Generational gc

What is generational gc? Why do generational gc?

Should we try to gc young objects or old objects? (Are young objects or old objects more likely to be garbage?)

When do young objects become old?

A problem with generational gc: what if an old object contains a pointer to a young object? Why is this a problem? How can this happen? What can we do about it?

Note: for immutable programming, can this happen?

Generational gc

Pointers from old objects to young objects are the main problem for generational gcs.

Possible solutions (what are these?):

a) Remembered list.

b) Remembered set.

c) Card marking.

d) Page marking.

What is the cost of collecting the youngest generation? (Assume that the youngest generation is 90% garbage.)

Incremental gc

What is incremental gc? Why are we interested in incremental gc?

What is concurrent gc? Why are we interested in concurrent gc?

What is the collector? The mutator?

Incremental gc

For an incremental gc, objects have three colors:

White: not visited yet by the gc.

Grey: the object has been visited, but the children have not.

Black: the object has been visited, and its children have too.

For copying gc, which objects are white? are grey? are black? What color are new objects created by the mutator?

Invariants for incremental gc:

1. No black object points to a white object.

2. Every grey object is known by the gc.

Incremental gc

Baker's algorithm is an incremental version of Cheney's copying gc.

It maintains an extra invariant:

3. The mutator never has a pointer to a white object.

How can Baker's algorithm achieve this?

Where does Baker's algorithm allocate new objects?

What is the extra cost in Baker's algorithm? What can be done about it?

GC and the memory hierarchy

GC has a bad reputation for thrashing caches - why?

Things we can do about this (Appel 21.6):

1. Try to get the most frequently used objects into L2 cache. (Why? How can a generational gc help?)

2. Try to allocate objects sequentially in memory. (Why? How can a copying gc help?)

3. Try to arrange for few cache conflicts for frequently used objects. (What does this mean? Why? How can a generational gc help?)

4. Prefetch memory for allocation. (What does this mean? Why? How can a copying gc help?)

5. Try for good locality of reference. (What does this mean? Why? How can a copying gc help?)

Summary

GC is a lot easier with support from the compiler! (For stack layout and object layout).

Generational gc avoids collecting the whole heap every gc.

Incremental gc spreads the gc load over time.

Good gcs play well with the memory hierarchy.

Next week: Final exam.