Heap Division for Garbage Collection

Simon Dardis

dardiss@tcd.ie

May 13, 2011

Overview

- GC constraints
- Heap division
- Example

Garbage collection constraints

The time taken for a collection cycle is proportional to size of the heap.

Collectors examine all thread stacks/global variables.

Problems with these constraints

Garbage collectors must examine all thread stacks to determine roots. This can be speed up by caching roots (Cheng 1998).

Stacks can be frequently revisited if a subset of an application's threads have high allocation rates.

Partitioning the main heap

Generational collectors can partition the heap but cannot collect an arbitrary generation.

Partitioning the main heap

Generational collectors can partition the heap but cannot collect an arbitrary generation.

Work done by Martin Hirzel (2002) can divide an application's heap into multiple partitions which may have dependencies on collection between them.

Partitioning the main heap

Generational collectors can partition the heap but cannot collect an arbitrary generation.

Work done by Martin Hirzel (2002) can divide an application's heap into multiple partitions which may have dependencies on collection between them.

In both cases the programmer has little control over how the heap is divided.

Programmer specified heap division

What if programmers could divide the heap into sections which are independently collected?

Programmer specified heap division

What if programmers could divide the heap into sections which are independently collected?

In a system where programmers can divide an application's heap in both space and memory management policy, they can control how garbage collection affects an application.

Programmer specified heap division - basics

An application's heap is divided into multiple *heap sections*.

A heap section has one or more threads associated with it and every thread must be associated with one heap section.

Programmer specified heap division - basics

An application's heap is divided into multiple *heap sections*.

A heap section has one or more threads associated with it and every thread must be associated with one heap section.

Message passing (with copying semantics) is used to share data between and within heap sections.

Each heap section is associated with a garbage collector.

Programmer specified heap division - message passing

Message passing is implemented as part of the underlying runtime for the application.

This allows the compiler or runtime to determine how to pass messages either pointer exchange, copying them or creating references that cross heap sections.

Programmer specified heap division - message passing

Message passing is implemented as part of the underlying runtime for the application.

This allows the compiler or runtime to determine how to pass messages either pointer exchange, copying them or creating references that cross heap sections.

Cross heap references can complicate some garbage collector designs since relocating an object requires the co-operation of two collectors.

Benefits

As each section is independent, collection work for a section is driven by the amount of data allocated by its associated thread(s).

Benefits

As each section is independent, collection work for a section is driven by the amount of data allocated by its associated thread(s).

The duration of the collection cycle and amount of data examined by a collector is proportional to the size of the heap section being collected.

Programmer specified heap division - example

An application's heap could be divided into sections for the GUI, error logging and application specific processing.

Programmer specified heap division - example

An application's heap could be divided into sections for the GUI, error logging and application specific processing.

A programmer could then use a concurrent collector for GUI and error logging for responsiveness and choose an appropriate collector for the application logic.

For a web browser, incremental or concurrent collectors could be used throughout. For an application like 3DS Max, a fast but simple mark-sweep collector could be chosen for rendering.

Target implementation

I am currently targeting Haskell as the source language.

New threads either belong to the heap section associated with their creator or get their own heap section.

Messaging style is similar to Erlang (asynchronous with mailbox, no ordering on receive unless from a single thread).

Summary

Heap division is a programmer driven mechanism that enables tuning of garbage collection for an application.

Heap division should increase collector efficiency since collection work is performed on a per section basis and the choice of collector can be based on the type of work the mutator performs.