Thread Local Heap Garbage Collection in a Simulated Runtime Environment

Marcel Dombrowski, Konstantin Nasartschuk, Kenneth B. Kent
University of New Brunswick
Faculty of Computer Science
marcel.dombrowski@unb.ca, kons.na@unb.ca, ken@unb.ca

Outline

- We developed a simulator for automated memory management, which allows prototyping of new Garbage Collection (GC) techniques.
- We implemented a Thread Local Heap (TLH) generational collector, where each thread has its own heap.
- Our prototype can easily be implemented in a Java Virtual Machine.

Motivation

- The development of new GC techniques always has a huge overhead as they need to be implemented in an existing system.
- To overcome this overhead we developed a simulator which only deals with automated memory management.
- This simulator is able to play back pre-recorded tracefiles, which contain object allocations and reference changes.
- This makes our simulator deterministic and thus suitable for comparison of different GC techniques.
- We implemented our own version of a TLH generational collector using this simulator.

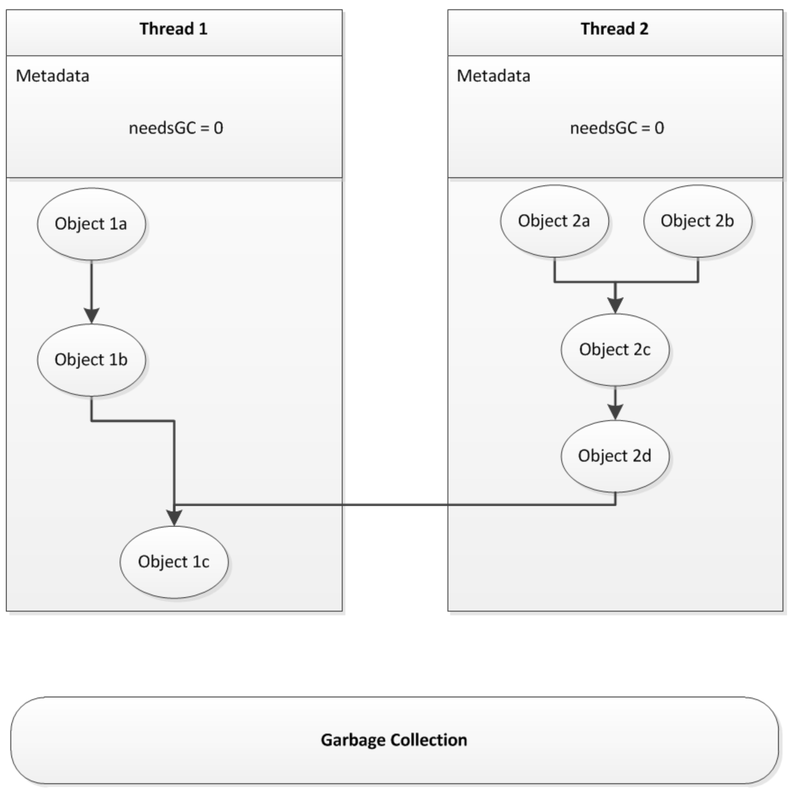

Figure 1: Object 1c escaped by being reachable from Thread 2.

Thread Local Heap GC

- Each thread has its own heap assigned to it, which it has to GC itself.
- An object escapes when it is reachable by more than one thread.
- What happens when all references within the thread to an escaped object are deleted?
  - We need to be able to detect if this object is still reachable from another thread.
- As soon as an object escapes, the original thread and the thread to which the object escapes to are put into its own group.
- If a GC occurs on a thread that is part of a group, all threads within this group are collected as well (worst case: all threads need to be collected).
- If all local references of an escaped object are deleted, the object will be collapsed (= moved) to a thread that still has a reference to it.
- If no reference is available the object can safely be deleted.

Results*

- We compared our results to a traditional generational collector using an artificially created tracefile.
- This tracefile contained 5 threads. We allocated heap sizes between 350 and 450 kB for the traditional generational GC and heap sizes between 80 and 100 kB for each thread for our TLH GC.
- On average 30-50% of the objects were escaped.
- We were able to successfully reduce the overall execution time. This is because smaller heaps take less time to GC and often GCs can be done during execution (actual improvements can only be seen when our approach would be implemented in e.g. a JVM).

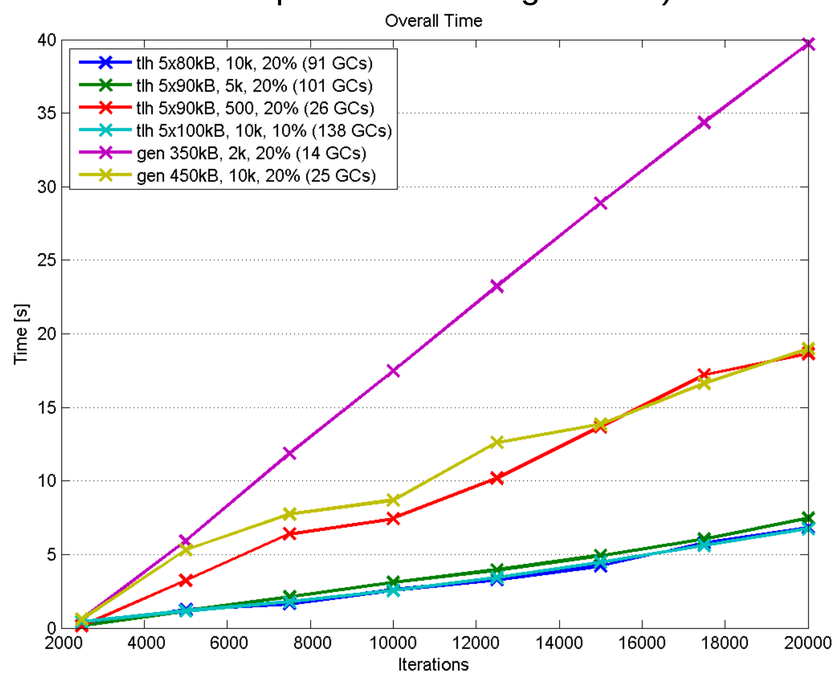

Figure 2: The overall runtime of our new technique compared to a traditional generational collector.

** Results unpublished at time of poster submission*