Exploiting the Generation Gap*

Steven K. Heller
December 30, 1987

Abstract

This paper presents several ways of optimizing a lifetime based garbage collector by exploiting the disparate natures of different regions. Ephemeral regions are small, contain a high percentage of garbage, and must be collected frequently and efficiently. Dynamic regions are large, contain relatively little garbage, and require infrequent collection. We customize optimizations for each region.

1 Introduction

Lisp Systems and other object oriented systems employ *garbage collection* [2] to recycle the space occupied by inaccessible objects. Lifetime based garbage collecting [6] and its implementations [7, 10, 3, 5, 8] are recent and leave room for optimization.

By organizing objects according to creation time, a lifetime based garbage collector can concentrate on young objects, those most likely to be garbage. The core of a lifetime based garbage collector involves tracking references from objects in older regions to objects in newer regions. Some schemes [7, 5, 8] track the references, some schemes [10] track the objects referred to, and some schemes [6, 3] break the references with an indirection.

Objects exist in three kinds of regions. Ephemeral regions are the youngest and smallest. Dynamic regions, the next oldest, are large and still subject to collection. A static region is considered the oldest; it is large but not subject to collection. There may be several ephemeral levels and several dynamic levels. Upon surviving a collection, objects can be promoted from one level to the next, through the ephemeral levels, the dynamic levels, and eventually to the static area. We assume that ephemeral objects are always promoted upon survival. The next section discusses custom made garbage collection strategies for individual regions.

*This paper describes research done at the Laboratory for Computer Science of the Massachusetts Institute of Technology, Cambridge, MA 02139. Funding for the Laboratory is provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-84-K-0099.

2 *Vive la Différence*

As pointed out in [6], different regions can have different collection strategies. We present a few ideas for collecting ephemeral regions, and an idea for collecting dynamic regions.

2.1 Copying Order Bows to Speed

Since objects are being copied, the collector should attempt to optimize the placement of objects to improve locality, or so says the lore. Locallity is important in both virtual memory systems and real memory plus cache systems. Moon's collector [7] uses pages as the blocking unit. But ephemeral objects never get paged out. If there is more than one ephemeral level, the *copying order makes no difference at all*, and the fastest one is best. The exception is the last ephemeral level, which promotes objects to dynamic status; it may be worth a bit of complexity at that point to improve locality. By simplifying the collection of the youngest ephemeral level, we can speed up our inner loop, and thereby the efficiency of our garbage collector as a whole.

2.2 Rates of Collection

The collection rates of the ephemeral levels can be set individually in much the same way that Baker controlled collection using the *Baker parameter, k*. [1] Suppose there are two ephemeral levels. The newest level can be collected to keep up with new object allocation, and the second level can be collected to keep up with survival from the first level. Evacuation traps, which prematurely evacuate objects, can be accounted for, slowing the collection rate even more. In this way collection can be spread out and delayed as much as possible, allowing better interactive response time, as well as more time for objects to die. A little complexity is cost effective in any but the first ephemeral level. In older ephemeral levels, for example, we can collect information concerning the dynamic behavior of the mutator. [3]

2.3 Cascading Leaves Spots

Before collecting the second ephemeral region, some collectors force the premature collection of the first ephemeral region. This collects inter-ephemeral-region garbage cycles, but has the unfortunate side effect of the double promotion of newly created objects. Collecting both regions in parallel, which takes more space, is an attractive alternative that gives new objects more time to die on their own while retaining the ability to collect the aforementioned cycles.

2.4 Old Objects Hang On

We should consider collecting dynamic regions using strategies other than copying, ones that take more time and less space. Dynamic regions contain a small fraction of garbage, and as a result require infrequent collection. Perhaps it's time for mark and sweep to emerge from the closet. The accessible fraction of the heap in older regions is likely to be high, so the size of the area that needs to be swept is not much larger than the size of the accessible area. Previously, mark-and-sweep strategies were dismissed in large address spaces due to the prohibitive overhead of sweeping. In this domain, however, they should be reconsidered. Several varieties of interleaved mark-and-sweep collectors have been developed [4, 9]. This is an interesting twist that has clear potential.

3 Conclusion

Virtual memory is an engineering combination of fast main memory and cheap secondary memory that approximates the speed of the former at the cost of the latter. Lifetime based garbage collection provides a similar multilevel scheme. Each level has its own characteristics, and is individually available for tailor-made optimizations.

References

[1] Henry G. Baker. List Processing in Real Time on a Serial Computer. *Communications of the ACM*, 21(4):280–294, April 1978.

[2] Jacques Cohen. Garbage Collection of Linked Data Structures. *ACM Computing Surveys*, 13(3):341–367, September 1981.

[3] Bob Courts. Obtaining Locallity of Reference in a Garbage-Collecting Memory Management System. November 1987. Internal TI Memo.

[4] Edsger W. Dijkstra, Leslie Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens. On-the-Fly Garbage Collection: An Exercise in Cooperation. *Communications of the ACM*, 21(11):966–975, November 1978.

[5] Richard Greenblatt. Greenblatt's Scheme for the LMI Lambda. December 1987. Personal Communication.

[6] Henry Lieberman and Carl Hewitt. A Real-Time Garbage Collector Based on the Lifetimes of Objects. *Communications of the ACM*, 26(6):419–429, June 1983.

[7] David A. Moon. Garbage Collection in a Large Lisp System. In *Symposium on Lisp and Functional Languages*, ACM, August 1984.

[8] Patrick G. Sobalvarro. Sobalvarro's Scheme for Lucid Common Lisp. December 1987. Personal Communication.

[9] Guy L. Steele. Multiprocessing Compactifying Garbage Collection. *Communications of the ACM*, 18(9):495–508, September 1975.

[10] David Ungar. Genaration Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm. In *Practical Programming Environments Conference*, pages 157–167, April 1984.