A Localized Tracing Scheme Applied to Garbage Collection

Yannis Chicha* and Stephen M. Watt

Department of Computer Science
University of Western Ontario
London Canada N6A 5B7

Abstract. We present a method to visit all nodes in a forest of data structures while taking into account object placement. We call the technique a *Localized Tracing Scheme* as it improves locality of reference during object tracing activity. The method organizes the heap into regions and uses trace queues to defer and group tracing of remote objects. The principle of localized tracing reduces memory traffic and can be used as an optimization to improve performance at several levels of the memory hierarchy. The method is applicable to a wide range of uniprocessor garbage collection algorithms as well as to shared memory multiprocessor collectors. Experiments with a mark-and-sweep collector show performance improvements up to 75% at the virtual memory level.

1 Introduction

Many algorithms require visiting all objects in a forest of data structures in a systematic manner. When there are no algorithmic constraints on the visiting order, we are free to choose any strategy to optimize system performance. This paper examines an optimization of object tracing to improve performance in a memory hierarchy. The basic idea is to delay the tracing of non-local objects and to handle them all together on a region-by-region basis. We call this a *localized tracing scheme* (LTS).

An LTS organizes its visit of the heap based partly on the graph of objects and partly on the location of objects. A consequence is that LTS can be memory hierarchy friendly, which means we are able to optimize visits of objects at different levels of the memory hierarchy, including cache, virtual memory and network. At one level, LTS can be used to reduce paging and keep object tracing in main memory as much as possible. At another level, as on-chip cache memory increases in size, LTS may be used to minimize traffic between cache and main memory. LTS may also be used in modern portable devices, where relatively large and slow flash memory cards extend the smaller and faster device memory.

Our LTS technique is based on dividing the heap into *regions* with a *trace queue* associated to each region to hold a list of objects to visit. Trace queues are the origin of the performance improvements displayed by the LTS. They are used to delay the tracing of remote objects, allowing tracing to concentrate on

* Present address: Teradyne SAS, 3 chemin de la Dhuy, 38240 Meylan, France.

N. Kobayashi (Ed.): APLAS 2006, LNCS 4279, pp. 323–339, 2006.

local objects. This enhances locality of reference by relying on object location, rather than object connectivity, to order tracing. The sizes of regions and trace queues are determined by the level of the memory hierarchy that we wish to optimize. For example, to obtain a cache-conscious algorithm, a region and the trace queues should be small enough to fit entirely in cache.

This idea may be applied to memory management, where reachable objects must be visited as part of garbage collection. Uniprocessor garbage collection is mature and offers satisfactory performance for many applications. In fact, garbage collection is now an integral part of the run-time for popular programming languages, such as Java and C#, which serve as the delivery platform for widely used applications. Improvements in garbage collection technology can therefore have impact on a broad user base.

We note that adding LTS to an existing collector is a relatively easy operation. This consideration is important in practice, where vendors are reluctant to make significant modifications to products' memory management for fear of introducing bugs. We have found it to be straightforward to modify two garbage collectors to use LTS, that of the Aldor programming language [1,2] run-time support and that of the Maple computer algebra system [3].

The impact of localizing tracing depends on the garbage collection method in use: Mark-and-sweep collectors first visit all live objects, marking them, and then sweep the memory area to recover unused space. Optimization of memory traffic during the *sweep* phase has been considered by Boehm [4]. We observe that memory hierarchy traffic can also be improved during the *mark* phase using LTS. Since objects do not move in memory with mark-and-sweep, the benefits of LTS are similar at each GC occurrence. Improvements of the overall GC time decrease when few objects are live. In this case, the mark phase is short and optimizations have a small impact.

Stop-and-copy garbage collectors can move objects to new locations at each GC occurrence. To do this, they must visit all live objects. Generational collectors [5] also use tracing because each generation is handled by a copying or mark-and-sweep collection algorithm. In these cases the tracing may be performed using LTS.

The rest of this paper is organized as follows. Section 2 describes a family of localized tracing algorithms. Section 3 gives an example to illustrate the LTS. Section 4 presents an informal proof of correctness for this algorithm. Section 5 details our experiments and results with the GC for the Aldor run-time environment. Section 6 explores advantages and drawbacks of the LTS in a multiprocessor environment. Section 7 discusses related work in various garbage collection settings. Section 8 suggests directions for future work and concludes the paper.

2 The Localized Tracing Scheme

2.1 Depth-First Tracing

We start by considering the usual recursive object tracing scheme. This will be useful for comparison with our local tracing algorithm.