Object Cache Locality in a Managed Runtime Environment

Marcel Dombrowski, Kenneth B. Kent, Michael Dawson, Dane Henshall
University of New Brunswick, IBM Canada
Faculty of Computer Science
{marcel.dombrowski|ken}unb.ca, {michael_dawson|dane_henshall}@ca.ibm.com

BACKGROUND

A managed runtime environment means that the programmers do not have to deal with the deallocation of memory, because it abstracts from the memory layer. The environment detects whether an object is dead in a garbage collect (GC) cycle. The Java Virtual Machine (JVM) is an example of a managed runtime environment.

These environments are usually operating system and hardware independent, which means that code written on one machine can be used on all machines that support this environment. The object layout in memory plays a crucial role in execution performance, as cache misses and page faults mean retrieval of memory from lower layers in the memory hierarchy (Figure 1).

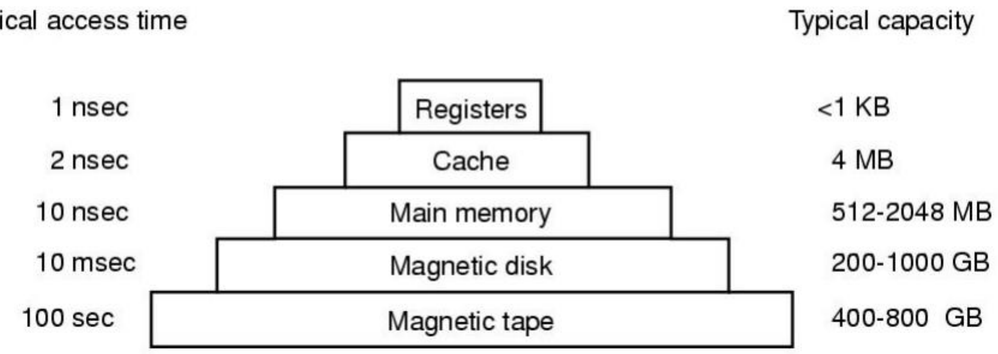

Figure 1: The memory hierarchy.

TECHNICAL APPROACH

Statistics have been gathered using different Java programs and benchmarks. These statistics include object temperature, popularity, memory distance, and object size. They will be used in order to extract knowledge and heuristics about object creation patterns.

The traversal of objects during a GC impacts the layout of objects in memory as it rearranges objects. Several naive and adaptive traversal approaches that relocate objects in memory depending on different metrics will be done. These include breadth first, depth first, and natural order traversal for the naive approaches and object hotness and heuristics based traversal for adaptive approaches.

Our GC simulator will be used in order to prototype these approaches and to conduct a feasibility study. A traversal evaluator has been written in order to determine the correctness of a traversal approach. A proof-of-concept implementation will then be done in the IBM JVM.

PROTOTYPING

Our simulator has been extended to include a real allocator which allocates the heap at the start up of the simulator and manages the available heap space. Interfaces have been extracted for our simulator that allow easy implementation of new allocators and collectors (Figure 2).

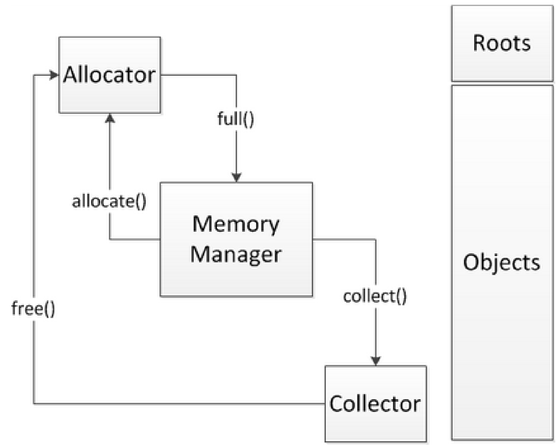

Figure 2: The architecture of our simulator.

A copying collector that uses a real heap has been implemented. For the copying collector we implemented a depth first and breadth first traversal and are currently implementing an object hotness based traversal (Figure 3). After that we will profile the simulator in order to determine the cache misses and page faults for several tracefiles. We extended the IBM JVM to allow tracing of objects. We use this extension in order to generate tracefiles from real world applications using the IBM JVM.

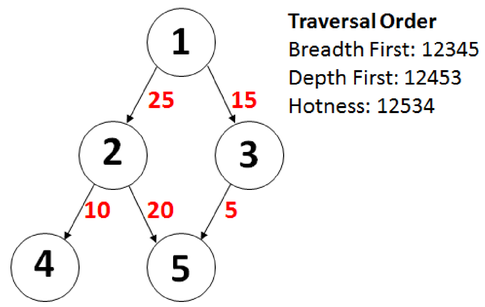

Figure 3: The traversal order for several approaches.

EVALUATION

Better cache locality may lead to better execution performance. We will prototype several traversal approaches in the simulator. If the prototyping of these approaches indicate an increase in performance or object locality the approaches will be applied to the IBM JVM for evaluation.

Several benchmarks will be used to determine the feasibility. Furthermore, we will discuss whether the overhead of adaptive approaches will outweigh the performance gain compared to naive approaches.

The results from the gathered statistics will be used to create a static ruleset for object creation prediction. This ruleset will then be evaluated by looking at the ratio of memory usage against cache misses.