Space-Efficient 64-bit Java Objects through Selective Typed Virtual Addressing

Kris Venstermans Lieven Eeckhout Koen De Bosschere
ELIS Department, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
{kvenster,leeckhou,kdb}@elis.UGent.be

Abstract

Memory performance is an important design issue for contemporary systems given the ever increasing memory gap. This paper proposes a space-efficient Java object model for reducing the memory consumption of 64-bit Java virtual machines. We propose Selective Typed Virtual Addressing (STVA) which uses typed virtual addressing (TVA) or implicit typing for reducing the header of 64-bit Java objects. The idea behind TVA is to encode the object's type in the object's virtual address. In other words, all objects of a given type are allocated in a contiguous memory segment. As such, the type information can be removed from the object's header which reduces the number of allocated bytes per object. Whenever type information is needed for the given object, masking is applied to the object's virtual address. Unlike previous work on implicit typing, we apply TVA to a selected number of frequently allocated and/or long-lived object types. This limits the amount of memory fragmentation. We implement STVA in the 64-bit version of the Jikes RVM on an AIX IBM platform and compare its performance against a traditional VM implementation without STVA using a multitude of Java benchmarks. We conclude that STVA reduces memory consumption by on average 15.5% (and up to 39% for some benchmarks). In terms of performance, STVA generally does not affect performance, however for some benchmarks we observe statistically significant performance speedups, up to 24%.

1 Introduction

Java is very popular on various computing systems, ranging from embedded devices to high-end servers. There are several reasons for the popularity of Java: its platform-independence as it relies on virtual machine (VM) technology, its object-oriented programming paradigm, its reliance on automatic memory management, etc. All of these factors contribute to the productivity enhancement of software development using Java.

Another well known phenomenon today is that the memory gap, *i.e.*, the gap between processor and memory speed continues to grow. As such, it is important to study techniques both in hardware and in software that help addressing the memory gap. On the hardware side, a multitude of techniques have been proposed to tackle the memory gap. Some examples are memory hierarchies, non-blocking caches, hardware prefetching, load address prediction, etc. On the software side, the memory gap can be tackled by reducing the amount of memory being consumed by applications, or by improving the data locality of an application, or by employing latency hiding techniques such as software prefetching, etc.

This paper focuses on reducing the memory consumption of 64-bit Java VM implementations. Our approach to reducing the memory consumption of 64-bit Java implementations is by proposing a Typed Virtual Addressing (TVA) mechanism that results in a space-efficient 64-bit Java object model. The TVA-aware VM that we propose removes the Type Information Block (TIB) pointer field from the object's header. Along with a number of other header layout modifications (that intelligently position the forwarding pointer in the object's header, as will be detailed in the paper) we reduce the object header size from 16 bytes to only 4 bytes for non-array objects, and from 24 bytes to only 8 bytes for array objects. The technology that enables removing the TIB pointer field is Typed Virtual Addressing which means that the object type is encoded in the object's virtual address. This is done by mapping all objects of the same type to the same contiguous memory segment. Accessing the TIB is then done by masking a number of bits from the object's virtual address, and using that as an offset in the TIB space that holds all the TIBs. Our proposal does not apply TVA to all object types but only to a selected number of types that are frequently allocated and tend to be long lived, hence the name Selective TVA (STVA). The reason is that applying TVA to all object types would result in too much memory fragmentation because of memory pages being sparsely filled with only a few objects.

The idea of typed addressing or implicit typing is not new. Typed addressing has been proposed in the past with proposals such as Big Bag of Pages (BiBOP), typed point-