SAFE AND EFFICIENT CLUSTER COMMUNICATION IN JAVA USING EXPLICIT MEMORY MANAGEMENT

Chi-Chao Chang, Ph.D.

Cornell University 2000

This thesis presents a framework for using explicit memory management to improve the communication performance of Java™ cluster applications. The framework allows programmers to explicitly manage Java communication buffers, called *jbufs*, which are directly accessed by the DMA engines of high-performance network interfaces and by Java programs as primitive-typed arrays. The central idea is to remove the hard separation between Java's garbage-collected heap and the non-collected memory region in which DMA buffers must normally be allocated. The programmer controls when a jbuf is part of the garbage-collected heap so that the garbage collector can ensure it is safely re-used or de-allocated, and when it is not so it can be used for DMA transfers. Unlike other techniques, jbufs preserve Java's storage- and type-safety and do not depend on a particular garbage collection scheme.

The safety, efficiency, and programmability of jbufs are demonstrated throughout this thesis with implementations of an interface to the Virtual Interface Architecture, of an Active Messages communication layer, and of Java Remote Method Invocation (RMI). The impact on applications is also evalu-

ated using an implementation of cluster matrix multiplication as well as a publicly available RMI benchmark suite.

The thesis proposes *in-place object de-serialization*—de-serialization without allocation and copying of objects—to further enhance the performance of RMI on homogeneous clusters. This optimization takes advantage of the zero-copy capabilities of network devices to reduce the per-object de-serialization costs to a constant irrespective of object size, which is particularly beneficial for large objects such as arrays. In-place de-serialization is realized using *jstreams*, an extension of jbufs with object I/O streams. Jstreams use the explicit memory management offered by jbufs to incorporate de-serialized objects into the receiving Java virtual machine without compromising its integrity, without restricting the usage of those objects, and without making assumptions about the underlying garbage collection scheme. The performance impact of jstreams on Java RMI and the benchmark suite is evaluated.