Space-Time Memory

U. Ramachandran[*] *R. S. Nikhil* *J. M. Rehg* *R. H. Halstead Jr.*[†] *C. F. Joerg*
L. Kontothanassis *K. Knobe*

Compaq Computer Corporation
Cambridge Research Lab
One Kendall Square, Bldg 700
Cambridge, Ma 02139
e-mail: {*kishore, nikhil, rehg, halstead, cfj, kthanasi, knobe*}@crl.dec.com

The Problem

Emerging application domains such as interactive vision, animation, and multimedia collaboration display dynamic scalable parallelism. Due to their high computational requirements, they are ideal candidates for executing on parallel architectures. Being interactive in nature, “time" is an important attribute in such applications. In particular, they require the efficient management of “temporally evolving" data. For example, a stereo module in an interactive vision application may need the correspondingly timestamped camera images from multiple cameras to compute its output. Further, both the data structures as well as the producer-consumer relationships in such applications are dynamic and unpredictable at compile time. Runtime systems (to date) for parallel computing do not offer any significant support for the application programmer to express such temporal requirements.

The Solution

To address these problems, we have developed a parallel programming abstraction called **space-time memory (STM)** – a dynamic concurrent distributed data structure for holding time sequence data. This abstraction is well-suited to providing the common parallel programming requirements found in most interactive applications, namely, buffer management, inter-task synchronization, and meeting *soft* real-time constraints. The key construct in STM is the *channel*, which is a location-transparent collection of items indexed by time. STM has operations to dynamically create a channel, and for a thread to *attach* to and *detach* from a channel. Each

[*] Current Affiliation: College of Computing, Georgia Institute of Technology, e-mail: rama@cc.gatech.edu
[†] Current Affiliation: Curl Co., e-mail: rhh@curl.com

attachment is known as a *connection*, and a thread may have multiple connections to the same channel. The *put* and *get* operations allow a thread to transact with a channel to which it is connected to store and retrieve data items (which are uninterpreted sequences of bytes for STM, or linked data structures with a user-specified interpretation). STM imposes rules on thread virtual times and generation of timestamps for items on the channels. These times are then used by STM to do automatic garbage collection of channel items. The API provided by STM allows the application programmer to write applications as he/she would on an SMP. The runtime system takes care of making operations on the STM valid even if the underlying architecture does not provide hardware shared memory. Currently, STM runs on a cluster of Alpha SMPs (on Digital Unix) interconnected by Memory Channel, and an interactive multimedia application called *Smart Kiosk* has been implemented on top of it.

Justifications

There can be no question that interactive multimedia applications will impact the kinds of systems to be built in the future. The space of parallel programming support for interactive applications is sparse. The key contribution of this work is to relieve the application programmer from the burden of low-level synchronization and buffer management by providing a higher level programming abstraction, and letting the runtime system worry about an efficient implementation of this abstraction. DSM systems are too low level for programming such dynamic applications, and existing parallel programming languages do not offer the right kind of abstraction for applications to express their temporal requirements.