Remote Queues: Exposing Message Queues for Optimization and Atomicity

Frederic T. Chong
ftchong@lcs.mit.edu

Lok T. Liu
ltliu@cs.berkeley.edu

Shamik D. Sharma
shamik@cs.umd.edu

John D. Kubiatowicz
kubitron@lcs.mit.edu

Massachusetts Institute of Technology
University of California at Berkeley
University of Maryland at College Park

We introduce Remote Queues (RQ), a communication model that integrates polling with selective interrupts to support a wide range of applications and communication paradigms. We show that polling is desirable for a range of applications for both performance and atomicity. Polling enables optimizations that are essential for fine-grain applications such as sparse-matrix solution. Polling also improves flow control for high-level communication patterns such as transpose.

We use RQ to implement active messages, bulk transfers, and fine-grain applications on the MIT Alewife, Intel Paragon and Cray T3D using extremely different implementations of RQ. RQ improves performance on all of the machines, and provides atomicity guarantees that greatly simplify programming for the user. RQ also separates handler invocation from draining the network, which simplifies deadlock avoidance and multiprogramming.

We also introduce efficient atomicity mechanisms on Alewife to integrate polling with interrupts, and discuss how to exploit interrupts on Alewife and the Intel Paragon without forfeiting the atomicity and optimization advantages of RQ.

Active messages, as developed by von Eicken et al. [E+92], are a powerful primitive for fine-grain communication. In particular, they can integrate incoming message data into the receiver's computation by interrupting the receiver and executing a handler.

Unfortunately, to avoid the high cost of interrupts, polling for messages has emerged as the default communication model on machines such as the CM-5 [L+92]. For example, the Strata communication library is based entirely on active messages with polling [BK94]. We have evidence that polling is desirable even in the presence of fast interrupts, such as on the MIT Alewife machine. Polling provides several advantages over interrupts including new opportunities for optimization, precise control over incoming traffic that enables better flow control, and atomicity by default, which greatly simplifies correct parallel programming.

We introduce the Remote Queue model (RQ) as a communication model equivalent in power (in its simplest form) to active messages with polling; it is thus simpler than full active messages. In essence, we separate the *queuing* of messages from the *invocation* of handlers. This enables new optimizations, provides a clean atomicity model, and simplifies implementation and multiprogramming.

Remote Queues (RQ) are a simple model that provides a general abstraction for low-level communication. RQ consists of three basic elements. First, one or more named queues at a receiving node. Second, an enqueue operation which causes data from a sending node to be placed in a named queue at a receiving node. Third, a dequeue operation which removes data from a named queue for processing by the compute processor (or a co-processor). Figure 1 illustrates possible implementations of the three basic elements of RQ.

We use RQ to implement active messages, bulk transfers, and fine-grain applications on the Intel Paragon and Cray T3D using extremely different implementations of RQ. We also show how polling on the CM-5 and Alewife maps to the RQ model. RQ enables improved performance on all of the machines.

Many machines, however, rely upon message interrupts to provide protection, support system functions,

This work was done in collaboration with Prof. Eric Brewer, brewer@cs.berkeley.edu, who is supported in part by TITAN and Mammoth, NSF grants CDA-8722788 and CDA-9401156. Fred Chong and John Kubiatowicz are advised by Prof. Anant Agarwal supported in part by ARPA contract N00014-94-I-09885, in part by NSF Experimental Systems grant MIP-9012773, and in part by an NSF PYI Award to Prof. Agarwal. Lok Liu is advised by Prof. David Culler and supported by DOE grant DEFG03-94ER25206 and by TITAN/Mammoth. Shamik Sharma is advised Prof. Joel Saltz and supported by NASA grant NAG-11560, by ONR grant SC 292-1-22913, and by ARPA grant NAG-11485. CM-5 support is from Project SCOUT, ARPA contract MDA972-92-J-1032.
Contact: Fred Chong, Room 613, 545 Technology Square, Cambridge MA 02139. (http://www.ai.mit.edu/people/ftchong/)

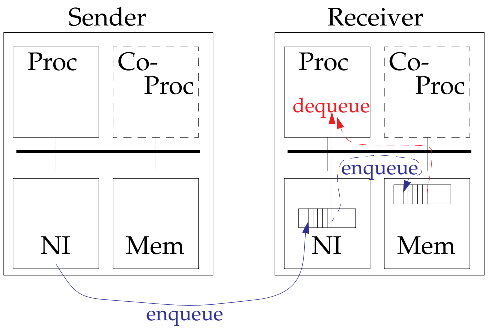

Figure 1: Implementations of the Remote Queuing (RQ) abstraction for low-latency communication with flexible synchronization on a range of parallel systems. Queues may reside in the Network Interface (NI) or main memory. The co-processor may also dequeue messages.

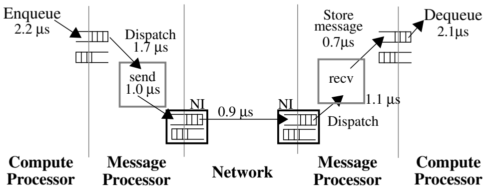

Figure 2: Implementation of RQ on the Paragon

and support advanced memory models. Message interrupts support both the global memory of the Cray T3D and the coherent shared memory of Alewife. These machines also rely upon interrupts to service system messages. Thus, we integrate RQ with support for selective interrupts, which allows full implementations of active messages. We introduce efficient atomicity mechanisms on Alewife to integrate RQ with interrupts, and discuss how to exploit interrupts on Alewife and the Paragon without forfeiting the atomicity advantages of RQ.

Figure 2 shows the costs for a example implementation of RQ on the Intel Paragon.

In summary, Remote Queues provide the following advantages:

- Clean support for atomicity. Code is atomic by default except for explicit dequeues. We can mix polling with interrupts by selectively using interrupts for system messages or infrequent user messages.

- RQ allows us to exploit knowledge, if any, about incoming messages. In particular, we can avoid handler dispatch and integrate reception directly into the computation loop, as was done for the sparse-matrix code.

- Explicit control of reception enables smoother flow control. In particular, we can precisely alternate sending and receiving, which leads to optimal performance for complex bulk communication patterns such as transpose.

- It is easy to integrate RQ with the rest of the communication system. We mixed RQ with DMA on the Paragon, T3D and Alewife. We also mixed RQ for user messages with selective interrupts for either system messages or unexpected user messages. We also showed that RQ enables new optimizations for complex patterns and strided transfers.

- We showed that RQ makes sense for coprocessors. It simplifies multiprogramming by separating the draining of the network from the handling of messages. This allows user- level handlers without restrictions, without deadlock, and without the need for strict gang scheduling.

- Finally, the benefits of RQ for coprocessors apply equally well to intelligent host interfaces in networks of workstations, which also require multiprogramming and integration of polling and interrupts.

A full paper on this research can be found in [BCL+95].

References

[BCL+95] Eric A. Brewer, Frederic T. Chong, Lok T. Liu, Shamik D. Sharma, and John Kubiatowicz. Remote queues: Exposing message queues for optimization and atomicity. In *1995 Symposium on Parallel Architectures and Algorithms*, Santa Barbara, California, July 1995. To appear.

[BK94] Eric A. Brewer and Bradley C. Kuszmaul. How to get good performance from the CM-5 data network. In *International Parallel Processing Symposium*, Cancun, Mexico, April 1994.

[E+92] Thorsten von Eicken et al. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, Queensland, Australia, May 1992.

[L+92] Charles E. Leiserson et al. The network architecture of the connection machine CM-5. In *Symposium on Parallel Architectures and Algorithms*, pages 272–285, San Diego, California, June 1992. ACM.