The FUGU Anti-Deadlock Strategy

Victor Lee, Kenneth Mackenzie,
Walter Lee, and Jon Michelson *
Fugu Group, MIT Lab for Computer Science
{wklee,kenmac,walt,jonm}@cag.lcs.mit.edu

1 Introduction

In concurrent computing, communication is important. Whether it is the communication between two separated uniprocessors or the communication between two nodes in a massively parallel supercomputer, it is essential to have a communication service that guarantees forward progress in delivering messages. Furthermore, in a multiprogrammed system, the system must be protected from misbehaved applications.

For a multiprocessor with internal reliable networks, a high-level anti-deadlock strategy is essential to fulfill the service guarantees. Typical anti-deadlock strategies in multiprocessors use the most straight-forward designs possible because it is often hard enough to get the basic system aspects correct. In designing Fugu, a multiuser multiprocessor, we want to identify the fundamental architectural features required to implement an anti-deadlock strategy and use them to build a cost-effective anti-deadlock strategy.

2 Finding the Fundamentals

Before we begin our search for the fundamental requirements, we first identify where deadlocks can occur in a multiprocessor. We know that deadlocks are caused by circular dependencies. In a system, circular dependencies can exist in any communication layer or in a combination of multiple layers. After a careful analysis, we identify three communication layers where circular dependencies occur. They are: the network layer, the protocol layer and the application layer.

Next, we study what different multiprocessors use to solve deadlocks at each layer. We find that most anti-deadlock strategies used in multiprocessors only focused on solving deadlocks in the network layer and the protocol layer. This is not entirely surprising to us because most previous-generation multiprocessors only support a single user or at best a crude form of multiprocessing with a simple scheduling scheme. There are few anti-deadlock strategies that actually cover application layer deadlocks. We identify three classes of approaches in these strategies: 1) transmitting all user level messages through a guarded operating system kernel layer [7, 5], 2) using a time-out mechanism to preempt applications that block the network for an extended period of time [1, 2, 8], and 3) providing a separate network buffer for each application and context-switching the network buffers when one application finishes its current time-slice or when it is deadlocked [3].

A study of these approaches leads us to identify the fundamental architectural features required to implement an system-wide application level anti-deadlock strategy. They are:

- A means to detect network congestion or deadlocks.
- A communication channel to allow system communications at all time.

However, this analysis only identifies the necessary features from the correctness point of view. It is not clear what performance requirements are needed for these features to provide an acceptable overall system performance. To explore this aspect, we are implementing an anti-deadlock strategy on the Fugu architectural simulator using these features and will measure the impact on the system performance with different design parameters.

3 The Fugu Strategy

The Fugu strategy uses a system timer to detect deadlocks. We recognize that a timer is not the only method to detect deadlocks, but it is the simplest and the most reliable method. The Fugu strategy resolves deadlocks using a two-phase approach: first passive then active. In the first, passive, phase, the Fugu strategy gives the network more system resources hoping the circular dependency can resolve itself. Past research shows that in most cases circular dependencies resolve if the network is given a little more resources [6]. Our implementation increases the network resources by allowing the network to overflow its buffer into the main memory. The main memory is backed up by external I/O devices. One use of the second network is to provide a communication channel for each node to reach the external resources when necessary. However, we believe that the system rarely needs to page out to external memory through the system network. Under

*We are advised by Anant Agarwal and supported in part by ARPA contract N00014-94-1-0985, by NSF Experimental Systems grant MIP-9012773, and by an NSF Presidential Young Investigator Award to Prof. Agarwal. Address: Room 632, 545 Technology Square, Cambridge MA 02139.

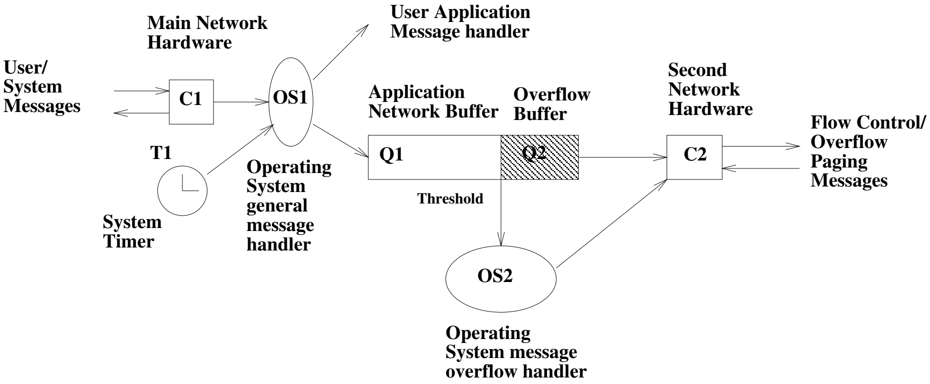

Figure 1: The Components in the Fugu strategy.

most circumstances, before the system runs out of physical memory, the Fugu strategy has entered the second phase which actively resolves the deadlock.

In the second phase, the Fugu strategy first gang-schedules the application that causes the deadlock. Gang-scheduling a distributed application reduces communication mismatches between its processes; thus, it should resolve any circular dependency caused by the mismatches. If gang-scheduling the application's processes does not work, the Fugu strategy then globally deschedules the problematic application except on the receiver node. This allows the application on the receiver node time to empty the network and to recover the system from the deadlock. Everytime the application is descheduled, the operating system will lower its scheduling priority. In this way, the poorly behaved application will not interfere with other running applications and it will eventually be completed. In the second phase, the second network is used to provide the necessary channel for system communication.

Figure 1 diagrams the Fugu strategy and depicts its operational dataflow. A user or a normal system message enters a node from the main network (C1). The operating system general message handler (OS1) then examines and dispatches the message. If the process that the message is dispatched to is not currently scheduled, or the message handler is busy or it has been timed out by the system timer (T1), OS1 buffers the message in the application's network buffer (Q1). When Q1 is filled to its threshold, the operating system invokes an overflow handler (OS2) and allocates an overflow buffer for the application. OS2 first attempts to gang-schedule the application. If gang-scheduling the application does not resolve the deadlock and the overflow buffer runs out of space, OS2 deschedules the application's sender processes. The second network (C2) is used to transmit OS2's "corrective" messages or to page out part of the message buffer if the node runs of physical memory.

4 Conclusion and Status

Analysis and simulations on the Fugu architectural simulator show that the Fugu anti-deadlock strategy can tolerate large transmission latency and small aggregated bandwidth in the second network without causing large system performance degradation. We are currently implementing the proposed strategy on the Fugu multiprocessor using a serial token ring second network [4, 9]. When the Fugu multiprocessor completes, we will be able to further evaluate the performance of the Fugu strategy.

[1] A. Agarwal et al. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA'95)*, pages 2–13, June 1995.

[2] D. Chiou et al. StarT-NG: Delivering Seamless Parallel Computing. In *Proceeding of the EURO-PAR'95 Conference*, August 1995.

[3] C. E. Leiserson et al. The Network Architecture of the Connection Machine CM-5. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, 1992.

[4] Ken Mackenzie et al. FUGU: Implementing Protection and Virtual Memory in a Multiuser, Multimodel Multiprocessor. Technical Memo MIT/LCS/TM-503, October 1994.

[5] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994.

[6] J. Kubiatowicz. The Anatomy of a Message Send. In *MIT Student Workshop on VLSI and Parallel Systems*, page 26, July 1992. Abstract of future paper on message interface.

[7] M. A. Blumrich et al. Two Virtual Memory Mapped Network Interface Designs. In *Proceedings of the Hot Interconnects II Symposium*, pages 134–142, Princeton, New Jersey, August 1994.

[8] S. K. Reinhardt et al. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21nd Annual International Symposium on Computer Architecture (ISCA'94)*, April 1994.

[9] Victor Lee. An Evaluation of Fugu's Network Deadlock Avoidance Solution. Master's thesis, Massachusetts Institute of Technology, June 1996.