Notes on "A Methodology for Implementing Highly Concurrent Data Objects"

JOSEPH P. SKUDLAREK
Mentor Graphics Corporation

1. INTRODUCTION

Recently, a novel pair of process synchronization instructions, *load_linked* and *store_conditional*, were introduced into the computing repertoire. These instructions were used to define two notable protocols for process synchronization: a lock-free and a wait-free protocol. These protocols do not require critical sections, and can be used for fault-tolerant processing. The lock-free protocol guarantees that at least one process will make progress; the wait-free protocol guarantees that all processes will make progress.

Herlihy [1993] presents and illustrates these concepts and ideas. However, there are a number of issues and errors concerning correctness, clarity, and completeness that were left unaddressed, and they are identified and resolved in Skudlarek [1994]. A summary of the issues and their resolution is presented below.

2. ISSUES AND RESOLUTIONS

The protocols described in Herlihy [1993] require a memory reference between the *load_linked* and *store_conditional* in order to validate the integrity or identity of the copied block. However, neither the MIPS [Kane and Heinrich 1992, p. A-64] nor the Alpha [Digital Equipment 1992, p. 4–9] architectures support such an interleaved memory fetch, and the protocols as presented can lock up. This persistent failure is different from the intermittent spurious failure discussed in Herlihy [1993, section 4.2]. One solution is to check and update both the pointer and the version id atomically, a technique similar to the one described in IBM [1980, p. 7–11; p. A-37–A-38] using the CDS (Compare Double and Swap) instruction. The Alpha architecture supports 64-bit *load_linked* and *store_conditional* instructions.

Herlihy [1993, section 4.2, p. 754] points out that special care in processing is required if storage is allowed to hold objects of different types—a second fetch is used to verify an object's identity. However, special care in storage layout is also required. In particular, the storage layout in which the check fields follow the sequential object allows the check fields to be overwritten arbitrarily, which can defeat the verification of object identity. Putting the

Author's address: Mentor Graphics Corporation, 8005 S. W. Boeckman Road, Wilsonville, OR 97070-7777; work email: jskud@wv.mentorg.com; personal email: jskud@teleport.com.

check fields at the front of the storage and requiring the check fields to be increased and never reset (even across object type changes) provide a solution for this problem.

The lock-free protocol adds just two check fields to the sequential object and avoids copying the check fields when block copying the concurrent object. The wait-free protocol requires a per-object response array which must remain synchronized with the sequential object. Yet the code given in Herlihy [1993, Figure 10, p. 761] copies only the sequential object. This problem is solved if the sequential object is redefined to include the response array for the wait-free protocol.

Skudlarek [1994] has more complete explanations as to why the double comparison avoids races in the wait-free protocol, and why the wait-free protocol requires operations which are totally defined for every valid object state.

A number of minor errors and issues, mostly typographical, are also identified and corrected in Skudlarek [1994]. For example, Herlihy [1993, Figure 2, line 11, p. 753] should be changed from

```
old_pqueue->check[1]++;   /* mark consistent */
```

to

```
new_pqueue->check[1]++,   /* mark consistent */
```

Herlihy [1993] assumes a *strongly ordered* memory model, in which all loads and stores are reflected in memory in the order they were issued by the processor; yet this is becoming a less common memory model. The standard memory model for recent SPARC Architectures is *total store order* (in which loads can occur ahead of stores), not *total access order—a strongly ordered* memory model is not provided. See SPARC [1992] and Weaver and Germond [1994].

ACKNOWLEDGMENTS

A number of people, including John F. Reiser, Michael T. Y. McNamara, and the referees, were very helpful in the preparation of this article. Maurice Herlihy, in his initial article, provided the foundation which serves as the context for these notes.

REFERENCES

DIGITAL EQUIPMENT 1992. *Alpha Architecture Handbook/Preliminary*. Digital Equipment Corporation, Maynard, Mass.

HERLIHY, M. P. 1993. A methodology for implementing highly concurrent data objects *ACM Trans. Program. Lang. Syst. 15*, 5 (Nov.), 745–770.

IBM 1980. *IBM System/370 Principles of Operation. 7th ed. GA22-7000-6*. IBM Corporation, Poughkeepsie, N.Y.

KANE, G. AND HEINRICH, J. 1992. *MIPS RISC Architecture*. Prentice-Hall, Englewood Cliffs, N.J.

SKUDLAREK, J. P. 1994. Remarks on a methodology for implementing highly concurrent data objects. *ACM SIGPLAN Not. 29*, 12 (Dec.), 87–93.

SPARC 1992. *The SPARC Architecture Manual, Version 8*. Prentice-Hall, Englewood Cliffs, N.J.

WEAVER, D. L. AND GERMOND, T. (Eds.) 1994. *The SPARC Architecture Manual, Version 9*. Prentice-Hall, Englewood Cliffs, N.J

Received June 1994; revised November 1994; accepted November 1994