Can Parallel Data Structures Rely on Automatic Memory Managers?

Erez Petrank
Dept. of Computer Science
Technion, Israel
erez@cs.technion.ac.il

ABSTRACT

The complexity of parallel data structures is often measured by two major factors: the throughput they provide and the progress they guarantee. Progress guarantees are particularly important for systems that require responsiveness such as real-time systems, operating systems, interactive systems, etc. Notions of progress guarantees such as lock-freedom, wait-freedom, and obstruction-freedom that provide different levels of guarantees have been proposed in the literature [4, 6]. Concurrent access (and furthermore, optimistic access) to shared objects makes the management of memory one of the more complex aspects of concurrent algorithms design. The use of automatic memory management greatly simplifies such algorithms [11, 3, 2, 9]. However, while the existence of lock-free garbage collection has been demonstrated [5], the existence of a *practical* automatic memory manager that supports lock-free or wait-free algorithms is still open. Furthermore, known schemes for manual reclamation of unused objects are difficult to use and impose a significant overhead on the execution [10].

It turns out that the memory management community is not fully aware of how dire the need is for memory managers that support progress guarantees for the design of concurrent data structures. Likewise, designers of concurrent data structures are not always aware of the fact that memory management with support for progress guarantees is not available. Closing this gap between these two communities is a major open problem for both communities.

In this talk we will examine the memory management needs of concurrent algorithms. Next, we will discuss how state-of-the-art research and practice deal with the fact that an important piece of technology is missing (e.g., [7, 1]). Finally, we will survey the currently available pieces in this puzzle (e.g., [13, 12, 8]) and specify which pieces are missing. This open problem is arguably the greatest challenge facing the memory management community today.

MSPC'12, June 16, 2012, Beijing, China.

Categories and Subject Descriptors

D.3.3 [**Language Constructs and Features**]: Dynamic storage management, Concurrent programming structures; D.3.4 [**Processors**]: Memory management (Garbage Collection); D.4.2 [**Storage Management**]: Garbage Collection

General Terms

Algorithms, Design, Performance, Reliability.

1. REFERENCES

[1] D. F. Bacon, P. Cheng, and V. Rajan. A real-time garbage collector with low overhead and consistent utilization. In *POPL*, 2003.

[2] F. Ellen, P. Fatourou, E. Ruppert, and F. van Breugel. Non-blocking binary search trees. In *PODC* , 2010.

[3] T. L. Harris. A pragmatic implementation of non-blocking linked-lists. In *DISC*, 2001.

[4] M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, 1991.

[5] M. Herlihy and J. E. B. Moss. Lock-free garbage collection for multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 3(3):304–311, 1992.

[6] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann, 2008.

[7] R. L. Hudson and J. E. B. Moss. Sapphire: Copying garbage collection without stopping the world. *Concurrency and Computation: Practice and Experience*, 15(3–5):223–261, 2003.

[8] G. Kliot, E. Petrank, and B. Steensgaard. A lock-free, concurrent, and incremental stack scanning mechanism for garbage collectors. *Operating Systems Review*, 43(3):3–13, 2009.

[9] A. Kogan and E. Petrank. Wait-free queues with multiple enqueuers and dequeuers. In *PPOPP*, 2011.

[10] M. M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. *IEEE Trans. Parallel Distrib. Syst.*, 15(6):491–504, 2004.

[11] M. M. Michael and M. L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *PODC*, 1996.

[12] E. Petrank, M. Musuvathi, and B. Steensgaard. Progress guarantee for parallel programs via bounded lock-freedom. In *PLDI*, 2009.

[13] F. Pizlo, E. Petrank, and B. Steensgaard. A study of concurrent real-time garbage collectors. In *PLDI* 2008.