Further Reading on Lock-Free Shared Data Structures for ESSCaSS 2011 Participants

Eric Ruppert
DisCoVeri Group
York University

The lectures at ESSCaSS will only scratch the surface of research on lock-free data structures. The following reading list describes where to find more information on the material that was presented, and gives some pointers to interesting papers that can serve as starting points in the literature for various topics.

Feel free to contact me for more information about these topics. (Contact information is on my web page www.cse.yorku.ca/∼ruppert.)

1 Introduction

Moore's Law originated in [39]. Borkar and Chien [10] discuss how improvements to microprocessors have been achieved in the past 20 years and some prospects for continued progress in the next 20 years, with an emphasis on parallelism.

Nir Shavit recently wrote a nice article [42] for the general reader discussing the new challenges that **multicore architectures** present to data structure design. A similar, shorter article by Herlihy [28] highlights the importance of designing lock-free data structures to take advantage of multicore architectures. (The quotation about short critical sections is from that article.)

There are several textbooks that give a good introduction to models of **distributed computing** and techniques used for designing algorithms for shared-memory systems. Both [8] and [30] include a number of lock-free data structures, in addition to being good all-around references on distributed computing. A survey by Moir and Shavit [38] can also serve as an introduction to **concurrent data structures**.

The **linearizability** correctness condition for concurrent accesses to shared objects was defined by Herlihy and Wing [31].

As a warning: In the lectures, I used the term **lock-free** to refer to the family of progress conditions that can be obtained without using locks, and I used the term **non-blocking** to refer to the property that eventually some operation makes progress (i.e., not all operations can be blocked forever). Unfortunately, if you read the distributed computing literature, these terms are sometimes used this way, and sometimes they are used interchangeably, and sometimes they are used with the meanings *reversed*.

The two major annual conferences that are most relevant to the topics discussed are the ACM's Symposium on Principles of Distributed Computing (**PODC**) and the International

Symposium on Distributed Computing (**DISC**). More information about these conferences may be found at www.podc.org and www.disc-conference.org.

2 Snapshot Objects

The notion of a **snapshot** object arose independently in three different papers [1, 2, 4]. The algorithms presented in the lectures reflect those in [1] most closely. A survey article on a number of snapshot implementations appears in [41].

Some more recent work on snapshots includes **partial snapshots**, which allow SCAN operations to read only some of the components of the snapshot object [7, 32] and **lower bounds** on the complexity of implementing snapshots [14].

3 Impossibility Results and Universal Constructions

Fischer, Lynch and Paterson [18] wrote a classic paper on the **impossibility** of solving consensus in an asynchronous message-passing system, even when only one process might fail. Their techniques were used by Herlihy [27] and by Loui and Abu-Amara [33] to prove that registers could not solve consensus (and hence could not implement many kinds of shared data structures). If you are interested in these impossibility results, or in lower bounds on the complexity of solving distributed problems, see the survey paper [17] for an introduction to this area.

Herlihy also provided the original **universal construction** in [27]. Numerous other improved universal constructions followed. A recent paper by Fatourou and Kallimanis [16] gives a good overview of the history of universal constructions and the current state of the art. (It also gives a new universal construction.)

Going beyond universal constructions, Shavit and Touitou defined software **transactional memory** [43]. This allows programmers to specify that chunks of code should be executed atomically. Transactions are a more complex notion than linearizable operations: transactions might be aborted (such a transaction reports to the invoker that the transaction failed). Even though aborted transactions should not affect any shared object's state, one must ensure that even the aborted transactions satisfy certain data consistency conditions. (See [23] for an introduction to transactional memory.) There have been attempts to design lock-free implementations of transactional memory, but they remain inefficient at present.

4 Stacks

The **stack** described in the lectures is by Treiber [45]. A good reference on the additional mechanisms needed to make lock-free stacks efficient is [26]. Once you know how to implement a stack, you might also be interested in how to implement a lock-free **queue**. A good starting point is Michael and Scott's queue [37]. These data structures are also discussed in the textbook [30].

5 Sets

I mentioned that there were several ways to provide lock-free implementations of a set (or dictionary) with INSERT, DELETE and SEARCH operations.

One of the best studied data structures in the lock-free setting is a sorted singly-linked **linked list**. This is a very inefficient data structure, since it requires linear searching, but it served as a testing ground for techniques for the design of lock-free data structures because of its simplicity. (A singly-linked list is also useful as a foundation for some simple data structures such as stacks and queues.) The first non-blocking list was given by Valois [46]. Other list implementations include [25, 35]. Fomitchev's M.Sc. thesis [19] provides a comparison of these implementations, and provides an implementation with better worst-case complexity. For a more recent paper on this topic, see [11].

Lock-free **skip lists** were developed by several researchers [19, 20, 44]. A similar implementation is now part of the Java standard library (java.util.concurrent.ConcurrentSkipListMap).

A few lock-free implementations of **hash tables** are also known [35, 40].

The **binary search tree** implementation that was presented is from [15]. It uses some ideas that are similar to Barnes's cooperative technique [9]. The experimental work on the search trees is from [12].

6 And Beyond...

These are a some aspects of lock-free data structures that I did not have time to cover in the lectures.

There are a few other kinds of data structures that have lock-free implementations, including **union-find** data structures [3] and **priority queues** [13].

There are various attempts to "optimize the common case". For example, even though there may be many processes in a system, it may be unlikely that they will all be accessing the same shared object at exactly the same time. Thus, it would be nice if the time required to perform an operation on the shared object depends on the number of processes concurrently accessing it, rather than the total number of processes in the system. Such an implementation is called **adaptive**. (See, for example, [6].) An even more extreme approach is to optimize the fast path: that is, to minimize the time required to perform an operation when an operation runs by itself.

A related concept is **obstruction-freedom**. Stronger progress conditions are harder to guarantee. So it is natural to ask whether there might be some progress guarantee that is weaker than the non-blocking property and easier to implement, but still strong enough to be useful. One candidate for such a progress condition is obstruction-freedom, which requires that an operation completes if it is permitted to take enough steps without interruption from other processes [29]. This progress property becomes useful especially when it is combined with a contention manager.

Memory management is a crucial problem for lock-free implementations. It can be quite tricky to determine whether it is safe to deallocate the memory used for a shared

object, because one must be sure that no other process is still holding a pointer to it. Reusing the same memory later can give rise to ABA problems too. Thus, performing lock-free garbage collection in an efficient way is a challenging problem. One powerful technique for accomplishing this is the use of hazard pointers [36], but it can be fairly complicated and will not always be applicable. See [21] for an example of an alternative approach using a mark-and-sweep garbage collection algorithm.

All of the algorithms discussed in these lectures were deterministic. **Randomization** is known to help a great deal in distributed computing. For example, no deterministic algorithm can solve consensus using registers, but there are efficient randomized algorithms that do. (See [5] for a survey of some randomized consensus algorithms.) A recent paper [22] takes an interesting look at the foundation of randomized implementations of shared objects.

Some cute theoretical questions include whether lock-free data structures can be implemented **anonymously** [24] or in systems with an **infinite** number of processes [34].

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, September 1993.

[2] James H. Anderson. Composite registers. *Distributed Computing*, 6(3):141–154, April 1993.

[3] Richard J. Anderson and Heather Woll. Wait-free parallel algorithms for the union-find problem. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 370–380, 1991.

[4] J. Aspnes and M. Herlihy. Wait-free data structures in the asynchronous PRAM model. In *Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures*, pages 340–349, 1990.

[5] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[6] Hagit Attiya and Arie Fouren. Algorithms adapting to point contention. *Journal of the ACM*, 50(4):444–468, July 2003.

[7] Hagit Attiya, Rachid Guerraoui, and Eric Ruppert. Partial snapshot objects. In *Proc. 20th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 336–343, 2008.

[8] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics, second edition.* Wiley, 2004.

[9] Greg Barnes. A method for implementing lock-free data structures. In *Proc. 5th ACM Symposium on Parallel Algorithms and Architectures*, pages 261–270, 1993.

[10] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, May 2011.

[11] Anastasia Braginsky and Erez Petrank. Locality-conscious lock-free linked lists. In *Proc. 12th International Conference on Distributed Computing and Networking*, pages 107–118, 2011.

[12] Trevor Brown and Joanna Helga. Non-blocking k-ary search trees. Technical Report CSE-2011-04, Department of Computer Science and Engineering, York University, 2011.

[13] Kristijan Dragičević and Daniel Bauer. A survey of concurrent priority queue algorithms. In *Proc. 22nd IEEE International Parallel and Distributed Processing Symposium*, pages 1–6, 2008.

[14] Faith Ellen, Panagiota Fatourou, and Eric Ruppert. Time lower bounds for implementations of multi-writer snapshots. *Journal of the ACM*, 54(6), December 2007.

[15] Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel. Non-blocking binary search trees. In *Proc. 29th ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2010.

[16] Panagiota Fatourou and Nikolaos D. Kallimanis. A highly-efficient wait-free universal construction. In *Proc. 23rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 325–334, 2011.

[17] Faith E. Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3):121–163, 2003.

[18] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[19] Mikhail Fomitchev. Lock-free linked lists and skip lists. Master's thesis, York University, 2003. Available from `www.cse.yorku.ca/~ruppert/Mikhail.pdf`.

[20] Keir Fraser. Practical lock-freedom. Technical Report UCAM-CL-TR-579, University of Cambridge, Computer Laboratory, February 2004.

[21] Hui Gao, Jan Friso Groote, and Wim H. Hesselink. Lock-free parallel and concurrent garbage collection by mark&sweep. *Science of Computer Programming*, 64(3):341–374, 2007.

[22] Wojciech Golab, Lisa Higham, and Philipp Woelfel. Linearizable implementations do not suffice for randomized distributed computation. In *Proc. 43rd ACM Symposium on Theory of Computing*, pages 373–382, 2011.

[23] Rachid Guerraoui and Michał Kapałka. *Principles of Transactional Memory*. Morgan&Claypool, 2010.

[24] Rachid Guerraoui and Eric Ruppert. Anonymous and fault-tolerant shared-memory computing. *Distributed Computing*, 20(3):165–177, 2007.

[25] Timothy L. Harris. A pragmatic implementation of non-blocking linked-lists. In *Proc. 15th International Conference on Distributed Computing*, volume 2180 of *LNCS*, pages 300–314, 2001.

[26] Danny Hendler, Nir Shavit, and Lena Yerushalmi. A scalable lock-free stack algorithm. *Journal of Parallel and Distributed Computing*, 70(1):1–12, January 2010.

[27] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.

[28] Maurice Herlihy. Technical perspective: Highly concurrent data structures. *Communications of the ACM*, 52(5):99–99, May 2009.

[29] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proc. 23rd IEEE International Conference on Distributed Computing Systems*, pages 522–529, 2003.

[30] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann, 2008.

[31] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, July 1990.

[32] Damien Imbs and Michel Raynal. Help when needed, but no more: efficient read/write partial snapshot. In *Proc. 23rd International Conference on Distributed Computing*, volume 5805 of *LNCS*, pages 142–156, 2009.

[33] Michael C. Loui and Hosame H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. In Franco P. Preparata, editor, *Advances in Computing Research*, volume 4, pages 163–183. JAI Press, Greenwich, Connecticut, 1987.

[34] Michael Merritt and Gadi Taubenfeld. Computing with infinitely many processes. In *Proc. 14th International Conference on Distributed Computing*, volume 1914 of *LNCS*, pages 164–178, 2000.

[35] Maged M. Michael. High performance dynamic lock-free hash tables and list-based sets. In *Proc. 14th ACM Symposium on Parallel Algorithms and Architectures*, pages 73–82, 2002.

[36] Maged M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):491–504, 2004.

[37] Maged M. Michael and Michael L. Scott. Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. *Journal of Parallel and Distributed Computing*, 51(1):1–26, 1998.

[38] Mark Moir and Nir Shavit. Concurrent data structures. In *Handbook of Data Structures and Applications*. CRC Press, 2004.

[39] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[40] Chris Purcell and Tim Harris. Non-blocking hashtables with open addressing. In *Proc. 19th International Conference on Distributed Computing*, number 3724 in LNCS, pages 108–121, 2005.

[41] Eric Ruppert. Snapshots in shared memory. In *Encyclopedia of Algorithms*, pages 855–858. Springer, 2008.

[42] Nir Shavit. Data structures in the multicore age. *Communications of the ACM*, 54(3):76–84, March 2011.

[43] Nir Shavit and Dan Touitou. Software transactional memory. *Distributed Computing*, 10(2):99–116, 1997.

[44] Håkan Sundell and Philippas Tsigas. Fast and lock-free concurrent priority queues for multi-thread systems. *Journal of Parallel and Distributed Computing*, 65(5):609–627, 2005.

[45] R.K. Treiber. Systems programming: Coping with parallelism. Technical Report RJ 5118, IBM Almaden Research Center, 1986.

[46] John D. Valois. Lock-free linked lists using compare-and-swap. In *Proc. 14th ACM Symposium on Principles of Distributed Computing*, pages 214–222, 1995.