Brief Announcement: Closing the Complexity Gap Between Mutual Exclusion and FCFS Mutual Exclusion

Robert Danek
University of Toronto, Canada
rdanek@cs.toronto.edu

Wojciech Golab
University of Toronto, Canada
wgolab@cs.toronto.edu

ABSTRACT

We consider the worst-case remote memory reference (RMR) complexity of first-come-first-served (FCFS) mutual exclusion (ME) algorithms for N asynchronous reliable processes, that communicate only by reading and writing shared memory. We exhibit an upper bound of $O(\log N)$ RMRs for FCFS ME, which is tight, improves on prior results, and matches a lower bound for ME (with or without FCFS).

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory.

1. INTRODUCTION

Mutual exclusion (ME), proposed by Dijkstra, allows multiple processes to access shared hardware or software resources safely. First-Come-First-Served (FCFS) ME, defined by Lamport, additionally ensures that processes are granted access to the resource in a fair order. In a FCFS ME algorithm, a process first executes a *doorway* – a bounded piece of code whose execution fixes the FCFS order. A process then waits for its turn in the *waiting room*, accesses the shared resource in the *critical section* (CS), and releases the resource in the *exit protocol*. A single execution of all these is a *passage*. Between passages, a process lives in the *non-critical section* (NCS).

We measure the time complexity of a ME algorithm by counting the number (per passage) of remote memory references (RMRs), which are memory accesses that traverse the processor-to-memory interconnect (e.g., cache misses).

Our main result is the first FCFS ME algorithm for N processes that has RMR complexity $O(\log N)$, and belongs to the class of algorithms that use read and write operations only. Our algorithm is also *adaptive* to point contention (denoted k), which is the maximum number of processes simultaneously outside the NCS. That is, it has RMR complexity $\Theta(\min(k, \log N))$, which is optimal for ME algorithms (with or without FCFS) in the class under consideration [1, 2]. Prior algorithms either rely on stronger synchronization primitives, lack FCFS, or have suboptimal RMR complexity [1, 3]. Thus, we close the RMR complexity gap between ME and FCFS ME for the class of algorithms that only use reads and writes.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.
ACM 978-1-59593-989-0/08/08.

2. THE FCFS ME ALGORITHM

The high-level structure of our ME algorithm is shown below. In the doorway, a process receives a ticket from a wait-free ticket dispenser (line 1), which behaves like a modular counter but may give identical tickets to processes that execute it concurrently. After obtaining a ticket, the process enters the waiting room (lines 2–3). Here it adds itself to a priority queue (Q) ordered by ticket, and then waits for its turn to advance into the CS. After leaving the CS, the process removes itself from the priority queue (line 5), and then returns to the NCS.

```
1 ticket := ObtainTicket();                    // Doorway.
2 Q.Insert((p, ticket));            // Waiting room begins.
3 await until my ticket is the smallest one "in use";
4 CS;                                  // The critical section.
5 Q.Remove((p, ticket));
```

The main challenge in fleshing out the above high-level structure is twofold: (1) how to implement ObtainTicket and Q correctly using only $\Theta(\min(k, \log N))$ RMRs and bounded memory; and (2) how to implement line 3.

We deal with the first challenge in a straight-forward way. For ObtainTicket, we use a circular array of bits, indicating which tickets are in use (i.e., held by some process between line 1 and line 5). To locate the next free ticket, we use a modified binary search that is adaptive to point contention. As for Q, we use a sequential implementation protected by an *auxiliary lock* (i.e., an existing ME algorithm with the required RMR complexity).

We answer the second challenge by having each process record its presence immediately before line 1 using a wait-free set-like data structure. To determine if it has the smallest ticket at line 3, a process checks whether it is the head of Q *and* whether the set-like data structure is empty, indicating that no process with a potentially smaller ticket is executing between line 1 and line 2.

3. REFERENCES

[1] J. Anderson, Y.-J. Kim, and T. Herman. Shared-memory mutual exclusion: Major research trends since 1986. *Distributed Computing*, 2002.

[2] H. Attiya, D. Hendler, and P. Woelfel. Tight RMR lower bounds for mutual exclusion and other problems. In *Proc. STOC*, 2008.

[3] G. Taubenfeld. The black-white bakery algorithm and related bounded-space, adaptive, local-spinning and FIFO algorithms. In *Proc. DISC*, pages 56–70, 2004.