Correctness of Hierarchical MCS Locks with Timeout

Milind Chabbi  
Hewlett Packard Labs  
Palo Alto, CA  
milind.chabbi@hpe.com

Abdelhalim Amer  
Argonne National Laboratory  
Lemont, IL  
aamer@anl.gov

Shasha Wen, Xu Liu  
College of William and Mary  
Williamsburg, VA  
{swen, xl10}@cs.wm.edu

This manuscript serves as a correctness proof of the Hierarchical MCS locks with Timeout (HMCS-T) described in our paper [1] titled “An Efficient Abortable-locking Protocol for Multi-level NUMA Systems,” appearing in the proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.

HMCS-T is a very involved protocol. The system is stateful; the values of prior acquisition efforts affect the subsequent acquisition efforts. Also, the status of successors, predecessors, ancestors, and descendants affect steps followed by the protocol. The ability to make the protocol fully non-blocking leads to modifications to the next field, which causes deviation from the original MCS lock protocol both in acquisition and release. At several places, unconditional field updates are replaced with SWAP or CAS operations.

We follow a multi-step approach to prove the correctness of HMCS-T. To demonstrate the correctness of HMCS-T lock, we make use of the Spin [2] model checking. Model checking causes a combinatorial explosion even to simulate a handful of threads. First we understand the minimal, sufficient configurations necessary to prove safety properties of a single level of lock in the tree. We construct HMCS-T locks that represent these configurations. We model check these configurations, which proves the correctness of components of an HMCS-T lock. Finally, building upon these facts, we argue logically for the correctness of HMCS-T(n).

1. MINIMAL CONFIGURATION

We need to answer the following questions to design an HMCS-T lock configuration that is sufficient to exercise all possible thread interleaving in any arrangement:

- How many threads are sufficient?
- How many lock levels are sufficient?
- How many lock acquisitions per participant are sufficient?

To answer these questions, we build non-deterministic finite acceptors (NFAs) that capture the state transition for each shared variable. The shared variables are the status and next fields of a QNode and the tail pointer variable. The transitions of the status field of a root-level QNode are different from the transitions of the status field of a non-root-level QNode. Figure 1 and Figure 2, respectively, show the NFA for the status field of a root-level and a non-root-level QNode. Figure 3 shows the NFA for the next field of any QNode. The tail pointer variable can be either null or non-null, and it is less interesting in designing the HMCS-T verification configurations. Appendix A, B, and C describe the transition associated with every edge shown in Figure 1 and 2, respectively.

Node labels in Figure 1, 2 represent the field values in those states, and the subscripts distinguish the same values that bear different meanings in different contexts. Solid black edges represent the actions taken by a thread t owning the QNode under scrutiny. Dotted blue edges represent the actions taken by a predecessor p of t. Dotted red edges represent the actions taken by a successor s of t. Thick black edges represent beginning of a new acquisition effort by a thread t that owns the QNode. Any subsequent path formed only of solid black edges represents a sequence of actions taken by a same thread of execution. Since the first operation in any acquisition is SWAPing the status field, every new acquisition edge has a SWAPing as its sink. Green color filled node(s) represent the state(s) where the lock containing thread t has become the owner of the lock at that level.

The NFA provides the following key insights:

1. Three participants: Any edge can be traversed via a path starting at the start state that involves no more than a predecessor (dotted blue edge), self (black edge), and a successor (dotted red edge) in Figures 1, 2, and 3. Hence, three participants (a predecessor, self, and a successor) are sufficient to exercise all possible transitions that the status field of a QNode may go through.

2. Two rounds: Any edge can be traversed via a path starting at the start state that involves no more than two “begin acquisition” (thick black line) edges. Hence, two rounds of acquisitions on the same QNode are sufficient to exercise all possible transitions. This means, at least, one thread should try two acquisitions. The other two threads can perform one acquisition each to exercise all interweaving of the third thread that performs two acquisitions.

3. Three levels: The edge $C_1 \rightarrow W_4$ in Figure 2 demands that a thread $t_1$ to have acquired the lock at the current node q at level l and abandoned at an ancestor level and a different thread $t_2$, a peer of $t_1$ at a level $< l$, to have inherited the level l lock from $t_1$. Hence, there should, at least, be two threads at level $l - 1$, which can cause one of them (say $t_1$) to acquire locks at level $l - 1$ and l but timeout at level $l + 1$ and eventually grant the locks at level $l - 1$ and l to another thread (say $t_2$). Three levels, parent, current, and children are sufficient to exercise all possible transitions in a non-root-level QNode.

To elaborate on Property 1 and 2, we describe a few interesting transitions in Figure 1. The edge $U_2 \rightarrow U_3$ needs
Figure 1: NFA for a QNode status field in HMCS-T(1).

Figure 2: NFA for the status field of a non-root-level QNode.

Figure 3: NFA for a QNode next field in HMCS-T(n). There is no designated "lock acquired" node.

Figure 4: Legend for Figures 1, 2, and 3.
t having a predecessor to reach U_2 and then a successor to cause impatience during the release protocol to transition to U_3. The edge W_3 → R_2 needs t to have a predecessor to reach U_2 and then the second round of acquisition attempt by t to reach U_3 and then a successor to make t impatient in its release protocol to eventually make the successor update t's status to R_2. The edge R_2 → U_1 and edge R_2 → A_1 need at least two rounds of acquisitions by t and a successor s to reach R_2. The same successor s can act as a predecessor for edge R_2 → U_1 transition. Similarly, s can act as a predecessor leading to a timeout to cause edge R_2 → A_1 transition.

NFAs, unfortunately, do not capture an important safety property—mutual exclusion. An NFA is ill-defined if the ownership of a QNode is not exclusive, which can happen if another thread belonging to the same domain starts modifying a shared QNode. To check the mutual exclusion property, we exercise all possible thread interleaving in a model checking phase.

To exercise all states of the root-level lock we use a thread configuration shown in Figure 6. The thread under scrutiny will be subjected to two rounds of acquisitions and the other two threads perform one round of acquisition each. Since model checking will exercise all interleaving, the timeout value is immaterial.

To exercise all states of the non-root-level lock, we use a thread configuration shown in Figure 5. There are two threads at level 1, which can cause one of them (say t_1) to acquire the locks at level 1 and 2 but timeout at level 3 and eventually grant the ownership of locks at level 1 and 2 to another thread (say t_2). The presence of two threads at level 1, also causes the common ancestor X, the QNode under scrutiny at level l, to go through the necessary two rounds of acquisitions. The other two participants—a successor s, and a predecessor p at level l—perform only one round of acquisition each. The model checking does not require s and p to begin the protocol at the leaf level, which avoids exercising some non-interesting interleavings. Hence, we set up s and p without children. Note that such arrangement is for model checking only; the HMCS-T lock admits new acquisitions starting at the leaf level only. In total, we need 4 threads, 2 at level 1 sharing the parent X, and 3 (of which one would have ascended from 1) at level 2. The behavior at level 3 will be non-deterministic—either a successful acquisition or abandonment to simulate all possible transitions in X. Non-deterministic behavior is easy to exhibit in Spin.

The verification checks for the assertion that two threads are never simultaneously in the critical section for the configuration in Figure 5. This assertion ensures that the root-level lock ensures mutual exclusion to the critical section if each QNode is accessed by descendent threads in a mutually exclusive manner. For the configuration in Figure 5, we check that t_1 and t_2 never simultaneously acquire the level l - 1 lock and no two threads ever simultaneously acquire the level l lock. This assertion ensures that a non-root-level lock ensures mutual exclusion to its next level if each QNode is accessed by descendent threads in a mutually exclusive manner.

Additionally, the NFAs in Figure 1, 2, and 3 provide insights into the following key properties:

1. **Livelock Freedom**: There does not exist any cycle without at least one “new acquisition” edge. Hence, there cannot be perpetual state transitions (live lock) without user opting to start another round of lock acquisition.

2. **Starvation Freedom**: Every W_i node (beginning of a new acquisition) has a path to the lock owning state (U_i in Figure 1 and V_i and C_i in Figure 2), if it is not allowed to traverse any timeout edge. This implies, every thread that starts its acquisition process and does not timeout, eventually acquires the lock. The next field does not decide the lock ownership and hence ignored.

3. **Bounded Steps to Release**: There exists a finite-length solid-black edge path from lock owner state to another node η such that a new acquisition (black edge) effort can begin at η. This implies, 1) an acquired lock can be released in a bounded number of steps by the lock owner and 2) once the lock is released, the QNode can be subjected to another acquisition attempt immediately.

4. **Bounded Steps on Timeout**: Every node that is not source node of a new acquisition edge (black edge) has a solid-black edge path to the source of a timeout edge. This implies that in any state after starting an acquisition process if a timeout occurs, t can abandon the protocol in a bounded number of steps. Source nodes of new acquisition edges are precluded because one cannot start an abandonment without having started an acquisition.

5. **Deadlock Freedom**: Every node has a path (there is an ε path to itself) formed out of solid-black edges to a node from where a new acquisition can begin.

### 2. Correctness of HMCS-T(N)

To establish the mutual exclusion guarantee of HMCS-T(n), we take the following steps:

**Lemma 2.1** (Root level lock ensures mutual exclusion): A root-level lock ensures mutual exclusion if every root-level QNode is owned by a descendent in a mutually exclusive manner.

**Proof.** Verified by model checking a root-level lock with the configuration shown in Figure 5.

**Lemma 2.2** (Non-root level lock ensures mutual exclusion): A non-root-level lock admits mutually exclusive access to the next level lock if every QNode at that level is owned by a single descendent at a time.

**Proof.** Verified by model checking a non-root-level in an HMCS-T lock with the configuration shown in Figure 4.

**Fact 2.1** (Exclusive ownership of leaf-level node): Every QNode at leaf level is owned by a unique thread, and the ownership is never shared with any other thread.
Theorem 2.1 (HMCS-T ensures mutual exclusion:) HMCS-T(n) ensures mutual exclusion to the critical section it protects.

Proof. HMCS-T(n) is composed of a root-level lock and n − 1 non-root-level locks. Each level ensures mutual exclusion to the level above as long the threads from descendant levels (if any) accesses the shared QNode at the current level in a mutually exclusive manner. Assume HMCS-T(n) does not ensure mutual exclusion to the critical section. This means two threads t1 and t2 can simultaneously be in the critical section. Both t1 and t2 are either 1) peers at level n and hence compete for the root-level lock at level n, or 2) belong to the same domain and hence compete for a non-root-level lock at a level l ≤ n.

If t1 and t2 are peers at level n, they will enqueue, two different QNodes and compete for the root-level lock and by Lemma 2.1 only one of them can be in the critical section at a time. Hence, t1 and t2 cannot be peers at the root-level.

Now, t1 and t2 are either peers at level n or belong to the same domain at level l′ < n − 1. If t1 and t2 are peers at level n − 1, they will enqueue two different QNodes and compete for the non-root-level lock at level n − 1 and by Lemma 2.2 only one of them can own the level n − 1 lock ensuring the mutual exclusion between them. Hence, t1 and t2 cannot be peers at level n − 1.

Since HMCS-T(n) has only a finite number of levels, by extrapolation, t1 and t2 are either peers at the leaf level or share the same QNode at the leaf level. If t1 and t2 are peers at the leaf level, they will enqueue two different QNodes and compete for the non-root-level lock at the leaf level and by Fact 2.2 only one of them can own the leaf level lock ensuring the mutual exclusion between them. Hence, t1 and t2 must be sharing the same QNode at the leaf level. By Lemma 2.1 no two threads can share the same QNode at the leaf level, hence t1 = t2, which contradicts the assumption.

Hence, only one thread can be in the critical section in HMCS-T(n).

Theorem 2.2 HMCS-T(n) guarantees live-lock freedom, deadlock freedom, starvation freedom, bounded steps to release, and bounded steps on timeout.

Proof. HMCS-T(n) is composed of a root-level lock and n − 1 non-root-level locks. By Fact 2.2 and 2.3 every thread follows an ordered acquisition and release or abandonment protocol. Hence, each thread goes through a finite number of levels in any process. At each level, root or non-root, the NFA that a thread is subjected to for its QNode, ensures live-lock freedom, deadlock freedom, starvation freedom, bounded steps to release, and bounded steps on timeout if the QNode is accessed mutually exclusively by descendants that share the same ancestor QNode. By Theorem 2.1 each QNode is owned by a descendant thread in a mutually exclusive manner. Hence, by construction HMCS-T(n) ensures live-lock freedom, deadlock freedom, starvation freedom, bounded steps to release, and bounded steps on timeout.

APPENDIX

A. NFA FOR THE STATUS FIELD OF A ROOT-LEVEL QNODE

The status always starts in R1 state. All other states are transient; a correctly implemented HMCS-T(1) ought to revert the status of very QNode to R1 eventually. On a fresh acquisition in the R1 state of a QNode q, the initial SWAP on q.status moves it non-deterministically to either W1 (if there was a predecessor) or W2 (no predecessor).

If no predecessor, the thread t updates q.status to U1 (edge W2 → U1). In U1, if t has a successor s that has already advertised itself with q.next or there is no successor, t releases the lock and updates q.status to R1 (edge U1 → R1). In U2, if t leaves due to timeout because a successor s has not updated q.next, the NFA transitions into state U3 (edge U1 → U3). In U3, if s advertises itself and recycles q.status, the NFA transitions to R3 (edge U3 → R1). In U3, if t attempts to re-acquire the lock, it will SWAP q.status to W4 (edge U3 → W4). If t times out in W4 while waiting for it to become R1, it reverts the state back to U3 (edge W4 → U3). In W4, if s advertises itself and recycles q.status, the NFA transitions to R2 (edge W4 → R2).

In W1, a predecessor may pass the lock to the waiting thread t updating q.status to U1 (edge W1 → U1). If t times out in W1, it updates the state to A1 (edge W1 → A1). In A1, a predecessor p may move the status to U2 (edge A1 → U2). In A1, any attempt by t to re-acquire the lock reverts the state to W1 (edge A1 → W1). In U2, if p manages to successfully release the lock, it will eventually transition q.status to R1 (edge U2 → R1). In U2, if p times out (impatient) waiting for a successor delayed in updating q.next, the NFA transitions to U3 (edge U2 → U3). In U3, any attempt by t to re-acquire the lock moves the state to W1 (edge U3 → W1). If t times out in W1, it reverts the state to U3 (edge W1 → U3). In W1, either a predecessor may update the state to recycled R2, or an impatient predecessor may time out and a successor may update the state to recycled R2 (edge W3 → R2).

In R2, t will requeue the QNode and it may acquire the lock via transition to U1 either because it has no predecessors or a predecessor passed the lock (edge R2 → U1). In R2, after enqueuing the node, if t times out waiting for the lock, it will transition to A1 (edge R2 → A1).

B. NFA FOR THE STATUS FIELD OF A NON-ROOT-LEVEL QNODE

We now describe the state diagram for the status field of a non-root-level QNode. The status always starts in R1 state. All other states are transient, a correctly implemented non-root-level ought to revert the status of very QNode to R1 eventually.
ally. On a fresh acquisition in the R₁ state of a QNode q, the initial SWAP on q.status moves it non-deterministically to either W₁ (if there was a predecessor) or W₂ (no predecessor).

If no predecessor, the thread t updates q.status to C₁ (edge W₂ → C₁). IN C₁, if t has a successor s that has already advertised itself with q.next or there is no successor, t releases the lock and updates q.status to R₁ (edge C₁ → R₁). In C₁, if t leaves due to timeout because a successor s has not updated q.next, t leaves q by updating its status to P₂ (edge C₁ → P₂). In P₂, if s advertises itself and recycles q.status, the NFA transitions to R₁ (edge P₂ → R₁). In P₂, if t attempts to re-acquire the lock, it will SWAP q.status to W₁ (edge P₂ → W₁). If t times out in W₁, while waiting for it to become R₁, it reverts the state back to P₂ (edge W₁ → P₂). In W₁, if s advertises itself and recycles q.status, the NFA non-deterministically transitions to either R₁ (edge W₅ → R₁, if it finds no predecessor by the time t re-enqueues the node) or to R₂ (edge W₅ → R₂, if a predecessor is present by the time t re-enqueues the node).

In R₂, t will reenqueue the QNode and it may inherit the global lock (transition to V₁, edge R₂ → V₁) or inherit only lock prefix (transition to P₁, edge R₂ → P₁) from one of its predecessors. In R₂, t may timeout and abandon while waiting for the lock (edge R₂ → A₁).

C. NFA FOR THE NEXT FIELD OF A QNODE

We now describe the state diagram for the next field. The next field starts with a null value in state 0₁. At the beginning of an acquisition, thread t transitions to 0₂, where the value of the next field remains unchanged from before (edge 0₁ → 0₂). If t finishes relinquishing the lock, the state reverts to 0₁ (edge 0₂ → 0₁). This transition can happen either by t itself (black solid edge) or after t has abandoned, which case a predecessor may act on t's behalf (blue colored dotted edge).

If a successor enqueues and advertises itself with a legal QNode pointer value S, NFA transitions to S₁ (edge 0₂ → S₁). t may successfully acquire the lock and release, which leaves it in S₁. t may timeout and abandon, which leaves it in S₁ and subsequent attempts to acquire by t will leave it in S₁ until a predecessor marks the QNode for recycling at which point t resets the next pointer to null just before enqueuing (edge S₁ → 0₂). In S₁, if t times out, a predecessor, may reuse the next field to remember the predecessor on its forward journey to find a waiting successor (edge S₁ → P₁). In S₁, if t attempts to re-acquire, it will wait and possibly timeout (edge S₁ → S₁). In P₁, once a predecessor has recycled the QNode, t will reset the next pointer to null and re-enqueue (edge P₁ → 0₂). In P₁, if t attempts to re-acquire, it will wait and possibly timeout (edge P₁ → P₁). In 0₂, if t timeouts during release waiting for the successor to update the next pointer, t writes M₁ (edge 0₂ → M₁). If t times out during acquire in 0₂, a predecessor may trigger the edge 0₂ → M₁ transition. In M₁, if t attempts to re-acquire, it will wait and possibly timeout (edge M₁ → M₁) until the node is recycled by the successor (edge M₁ → S₁).

D. REFERENCES

[1] M. Chabbi, A. Amer, S. Wen, and X. Liu. An Efficient Abortable-locking Protocol for Multi-level NUMA Systems. In Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP ’17, New York, NY, USA, 2017. ACM.

[2] G. J. Holzmann. The Model Checker SPIN. IEEE Transactions on Software Engineering — Special issue on formal methods in software practice, 23(5):279–295, May 1997.