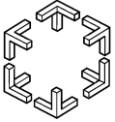



Distributed Computing and Systems
Chalmers university of technology



Lock-Free Skip List


Håkan Sundell
Philippas Tsigas



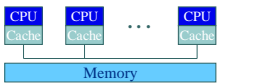
Outline

- Shared Memory
- Synchronization Methods
- Memory Management
- Shared Data Structures
 - Dictionary
- Performance
- Conclusions

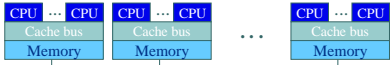
2



Shared Memory




- Uniform Memory Access (UMA)



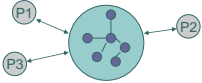
- Non-Uniform Memory Access (NUMA)

3




Synchronization

- Shared data structures needs synchronization!




- Accesses and updates must be coordinated to establish consistency.

4




Hardware Synchronization Primitives

- Consensus 1
 - Atomic Read/Write
- Consensus 2
 - Atomic Test-And-Set (TAS), Fetch-And-Add (FAA), Swap
- Consensus Infinite
 - Atomic Compare-And-Swap (CAS)
 - Atomic Load-Linked/Store-Conditionally

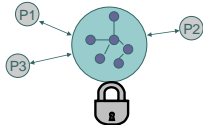


5



Mutual Exclusion

- Access to shared data will be atomic because of lock



- Reduced Parallelism by definition
- Blocking, Danger of priority inversion and deadlocks.
 - Solutions exists, but with high overhead, especially for multi-processor systems

6



Non-blocking Synchronization

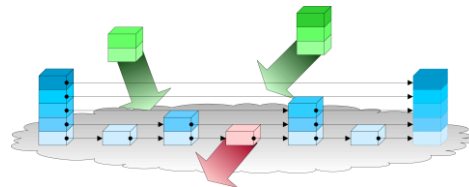
- Perform operation/changes using atomic primitives
- Lock-Free Synchronization
 - Optimistic approach
 - Retries until succeeding
 - Guarantees progress of at least one operation
- Wait-Free Synchronization
 - Always finishes in a finite number of its own steps
 - Coordination with all participants

7



Memory Management

- Dynamic data structures need dynamic memory management
 - Concurrent D.S. need concurrent M.M.!

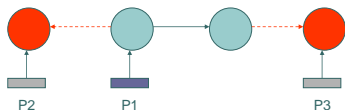


8



Concurrent Memory Management

- Concurrent Memory Allocation
 - i.e. malloc/free functionality
- Concurrent Garbage Collection
 - Questions (among many):
 - When to re-use memory?
 - How to de-reference pointers safely?



9



Lock-Free Memory Management

- Memory Allocation
 - Valois 1995: fixed block-size, fixed purpose
 - Michael 2004: Gidenstam et al. 2004, any size, any purpose
- Garbage Collection
 - Valois 1995, Detlefs et al. 2001: reference counting
 - Michael 2002, Herlihy et al. 2002: hazard pointers
 - Gidenstam, Papatriantafidou, Sundell and Tsigas 2005: hazard pointer + reference counting

10



Lock-Free Reference Counting

- De-referencing links
 - 1. Read the link contents, i.e. a pointer.
 - 2. Increment (FAA) the reference count on the corresponding object.
- What if the link is changed between step 1 and 2?
 - Solution by Detlefs et al:
 - Use DCAS on step 2 that operates on two arbitrary memory words. Retries if link is changed after step 2.
 - Solution by Valois et al:
 - The reference count field is present indefinitely. Decrement reference count and retries if link is changed after step 2.

11



Lock-Free Hazard Pointers (Michael 2002)

- De-referencing links
 - 1. Read the link contents, i.e. a pointer.
 - 2. Set a hazard pointer to the read pointer value.
 - 3. Read the link contents again; if not same as in step 1 then restart from step 1.
- Deletion
 - After deleted from data structure, put node on a local list.
 - When the local list reaches a certain size; scan all hazard pointers globally, reclaim memory of all nodes which address does not match the scan.

12

Lock-Free Memory Allocation

- o Solution (lock-free), IBM freelists:
 - Create a linked-list of the free nodes, allocate/reclaim using CAS

- Needs some mechanism to avoid the ABA problem.

13

Shared Data Structure: Dictionaries (Sets)

- o Fundamental data structure
- o Works on a set of <key,value> pairs
- o Three basic operations:
 - Insert(k,v): Adds a new item
 - v=FindKey(k): Finds the item <k,v>
 - v=DeleteKey(k): Finds and removes the item <k,v>

14

Randomized Algorithm: Skip Lists

- o William Pugh: "Skip Lists: A Probabilistic Alternative to Balanced Trees", 1990
- Layers of ordered lists with different densities, achieves a tree-like behavior

- Time complexity: $O(\log_2 N)$ – probabilistic!

15

New Lock-Free Concurrent Skip List

- Define node state to depend on the insertion status at lowest level as well as a deletion flag

- Insert from lowest level going upwards
- Set deletion flag. Delete from highest level going downwards

16

Overlapping operations on shared data

- o Example: Insert operation - which of 2 or 3 gets inserted?
- o Solution: Compare-And-Swap atomic primitive:

```

CAS(p:pointer to word, old:word, new:word):boolean
atomic do
  if *p = old then
    *p := new;
    return true;
  else return false;
  
```

17

Concurrent Insert vs. Delete operations

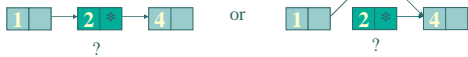
- o Problem: - both nodes are deleted!
- o Solution (Harris et al): Use bit 0 of pointer to mark deletion status

18



Helping Scheme

- Threads need to traverse safely



- Need to remove marked-to-be-deleted nodes while traversing – Help!
- Finds previous node, finish deletion and continues traversing from previous node



19



Lock-Free Skip List - Techniques Summary

- The Skip List is treated as layers of ordered lists
- Uses CAS atomic primitive
- Lock-Free memory management
 - IBM Freelists
 - Reference counting (Valois+Michael&Scott)
- Helping scheme
- Back-Off strategy
- All together proved to be linearizable



20



Lock-Free Skip List publications

- First publications in literature:
 - H. Sundell and P. Tsigas, "Fast and Lock-Free Concurrent Priority Queues for Multi-thread Systems", IPDPS 2003
 - H. Sundell and P. Tsigas, "Scalable and Lock-Free Concurrent Dictionaries", SAC 2004
- Later publications:
 - M. Fomitchev and E. Ruppert, "Lock-free linked lists and skip lists", PODC 2004
 - K. Fraser, "Practical lock-freedom", PhD thesis, 2004

21



New Lock-Free Skip List !

- The thread that fulfils the deletion of a node removes the next pointer when finished.
- Allows other threads to traverse through even marked next pointers.
- If not possible to traverse forward, go back to the remembered position on previous (upper) levels.
- Helps deletions-in-progress only when absolutely necessary.
- Works with a modified version of Michael's Hazard Pointer memory management!

22



Correctness

- Linearizability (Herlihy 1991)
 - In order for an implementation to be *linearizable*, for every concurrent execution, there should exist an *equal sequential execution* that respects the *partial order* of the operations in the concurrent execution

23



Correctness

- Define precise sequential semantics
- Define abstract state and its interpretation
 - Show that state is atomically updated
- Define linearizability points
 - Show that operations take effect atomically at these points with respect to sequential semantics
- Creates a total order using the linearizability points that respects the partial order
 - The algorithm is linearizable

24

Memory Consistency and Out-Of-Order execution

- Models on actual multiprocessor architectures: Relaxed Memory Order etc.

T_i $W(x,0)$ $W(x,1)$ $R(y)=0$ $R(y)=1$
 T_j $W(y,0)$ $R(x)=1$ $W(y,1)$
 T_k $R(x)=0$ $R(y)=1$ $R(x)=1$

- Must insert special machine instructions (memory barriers) to enforce stronger memory consistency models!

25

Experiments

- Experiment with 1-32 threads performed on Sun Fire 15K with 48 cpu's.
 - Each thread performs 20000 operations, whereof the first total 50-10000 operations are Insert's, remaining are equally randomly distributed over Insert, FindKey and DeleteKey's.
 - Fixed Skiplist maximum level of 10.
- Compare with implementations of other skip list-based dictionaries and a singly linked list by Michael, using same scenarios.
- Averaged execution time of 10 experiments.

26

