Concurrent Data Structures

Mark Moir and Nir Shavit

Sun Microsystems Laboratories

presented by

Simon Kranzer

Concurrency and Memory Management Seminar

WS 2010

University of Salzburg

Table of Contents

- The Authors
- Motivation
- Introduction
- Designing Concurrent Data Structures
- Shared Counters and Fetch-and-Φ
- Stacks and Queues
- Pools
- Linked Lists
- Hash Tables
- Search Trees
- Priority Queues

The Authors – Mark Moir

- Dr. Mark Moir
  - received the B.Sc.(Hons.) degree in Computer Science from Victoria University of Wellington, New Zealand in 1988, and the Ph.D. degree in Computer Science from the University of North Carolina at Chapel Hill, USA in 1996.
  - from August 1996 until June 2000, he was an assistant professor in the Department of Computer Science at the University of Pittsburgh.
  - in June 2000, he joined Sun Microsystems Laboratories, where he is now a Distinguished Engineer and the Principal Investigator of the Scalable Synchronization Research Group.

[2]

The Authors – Nir Shavit

- Dr. Nir Shavit
  - received the MS degree in Computer Science from Technion, Israel Institute of Technology in 1986, and the Ph.D. degree in Computer Science from the Hebrew University in Jerusalem, Israel in 1990
  - was a postdoctoral researcher at IBM Almaden, Stanford, and MIT, visiting faculty at MIT, and is a faculty member in the school of computer science at Tel-Aviv University.
  - is a researcher for Sun Microsystems Laboratories, working in the Scalable Synchronization Research Group.
  - is with Maurice Herlihy the joint recipient of the 2004 ACM/EATCS Gödel Price

[2]

Motivation

- In shared-memory multiprocessing multiple threads are executed concurrently
- Communication and synchronization is done via data structures in shared-memory
- Thus these data structures have to be
  - efficient
  - scalable
  - correct

[1]

Introduction

- Moir and Shavit provide an overview of the challenges of designing concurrent data structures
  and
- a summary of relevant work for some important data structures.
- Popular data structures have been chosen to illustrate key design issues

[1]

Designing Concurrent DS

- Concurrent DS Challenges
- Performance
- Blocking
- Nonblocking
- Measuring Complexity
- Proofing Correctness
- Locks, Barriers, Transactions

[1]

Concurrent DS Challenges

- Threads, executed concurrently on different processors/cores are influenced by
  - Scheduling
  - Page Faults
  - Interrupts
  - etc.
- Performance is influenced by
  - Memory layout
  - Processor layout
  - Layout of data in memory
- Hard to design and verify a correct DS implementation

[1]

Example: Shared Counter

- fetch-and-inc
  - Return value and increment counter by one
- Straight forward implementation does not comply with concurrency (bad interleaving)
  - x = counter;
    counter+=1;
    return x;
- The use of a mutex solves the problem but creates many more

[1]

Concurrent DS Performance

- Amdahls Law
  - n… Number of Proccessors
  - p… Fraction of job that can be parallelized

$$S = \frac{1}{1 - p + \frac{p}{n}}$$

- Example:
  If only 10% of an application **can not** be implemented in parallel speedup is only 5.3 on a machine with 10 processors

[1],[3]

Concurrent DS Performance

- Memory Contention
  - Multiple threads attempting to access the same location in shared memory
  - In a cache-coherent system this means heavy loads
- If data location is locked by a thread which is delayed (waits on I/O or whatever) all other threads have to wait too

[1]

Concurrent DS Blocking

- Memory contention can be reduced using a fine-grained locking scheme
  - Different parts of a DS can be accessed concurrently
- Operations can be spread out in time
  - Backoff algorithm
  - Combining trees

[1]

Concurrent DS Blocking

- Combining trees
  - Join concurrent operations
  - Winner transfers combined result/value to DS
  - Local spinning of losers in a cash-coherent multiprocessor system
  - Speedup of O(P/log(P)) with P as the number of threads
- Combining trees drawbacks
  - False sharing
  - Do not scale on low loads
  - Delays threads failed to combine

[1]

Concurrent DS Blocking

- Blocking DS can scale if there is a good balance between using enough blocking to maintain correctness, while minimizing blocking in order to allow concurrent operations to proceed in parallel

[1]

Concurrent DS Nonblocking

- Nonblocking progress conditions
  - Wait-freedom guarantees that an operation will complete after a finite number of its own steps not influenced by others
  - Lock-freedom guarantees that one of the concurrent operations completes after a finite number of steps
  - Obstruction-freedom means that an operation completes in a finite number of own steps after interference form other operations stopped

[1]

Concurrent DS Nonblocking

- Implementation of lock free fetch-and-inc
  - Atomic instructions provided by hardware or OS
    - CAS (Compare and Swap)
    - LL/SC (Load-linked/Store-conditional)
  - As CAS and LL/SC are universal they can be adopted to any DS if atomicity is provided
- Drawbacks
  - Sequential bottleneck
  - More difficult to handle because a lock can prevent other threads from interfering

[1]

Concurrent DS Measuring Complexity

- Idealized models do not reflect the real-world behavior of the DS because it is dominated from
  - Cost of contention
  - Cache behavior
  - Cost of universal synchronization
  - Arrival rates
  - Backoff delys
  - Memory layout, etc.
- Concurrent DS design is compared empirically by running them using micro-benchmarks

[1]

Concurrent DS Proofing Correctness

- As operations on sequential DS are executed one-at-a time correctness can be that the resulting sequence of operations respect sequential semantics
- For concurrent DS sequential consistency is a common condition
  - The total order preserves the order of operations executed by each thread

[1]

Concurrent DS Proofing Correctness cont.

- Linearizability
  - DS is sequentially consistent
  - Total ordering respects the real-time ordering among the executed operations
- Quiescent consistency
  - No real-time ordering
  - Every operation executed after a state without operations must be ordered after operations before that state

[1]

Concurrent DS Key Mechanisms

- Locks
  - Guarantee mutual exclusion
  - Spinlocks
  - Exponential backoff
  - Queuelocks
  - Abortable (queue)locks
  - Preemption-save locks
  - Reader-Writer locks
  - Group mutual exclusion

[1]

Concurrent DS
Key Mechanisms cont.

- Barriers
  - Collectively holds threads at a given point
  - Counter with number of threads
  - Spin on local memory
  - Diffusing computation tree
    - Threads are owners of nodes in a binary tree
    - Waiting for the arrival of their children
    - Root node releases all threads if all children are done

[1]

Concurrent DS
Key Mechanisms cont.

- Transactions
  - Treat sections code that access multiple memory locations as atomic
  - Relational databases
  - Hardware-based transactional memory
  - Software transactional memory

[1]

Shared Counters and Fetch-and-Φ

- Combining trees ✓
  - Blocking
- Counting Networks
  - Acyclic networks constructed from balancers
  - Tokens arrive at a balancer at arbitrary times and are output in a balanced way
  - Only capable of reduced class of operations like fetch-and-inc
  - Lineraizable only with drawbacks
  - Lock-free, quiescent consistent

[1]

Stacks and Queues

- Stacks
  - A concurrent stack is a data structure linearizable to a sequential stack that provides push and pop operations with the usual LIFO semantics
  - Lock-based implementation based on sequential linked lists using a top pointer and a global lock
    - Contention
    - Sequential bottleneck
  - Lock-based implementation using combining
    - Does not scale on low loads

[1]

Stacks and Queues

- Stacks cont.
  - Lock-free implementation using CAS and a single-linked list with top pointer
    - Sequential bottleneck (top pointer)
    - Faster than lock-based but does not scale under heavy load
- Queues
- Dequeues

[1]

Stacks and Queues

- Queues
  - A concurrent queue is a data structure linearizable to a sequential queue that provides enqueue and dequeue operations with the usual FIFO semantics
  - Lock-based implementation with separate locks for head and tail pointer of linked list
    - Concurrent dequeue and enqueue
    - Additional dummy element needed that head pointer never = tail pointer

[1]

Stacks and Queues

- Queues cont.
  - Lock-free CAS-based implementation with access to both ends of the queue using CAS in stead of locks
    - Dummy node
    - Operations can access already removed elements

[1]

Stacks and Queues

- Dequeues
  - A concurrent double-ended queue *is a linearizable concurrent data structure that* generalizes concurrent stacks and queues by allowing pushes and pops at both ends
  - Lock-based implementation as with queue
  - NO lock-free implementation with concurrent operations on both ends known

[1]

Pools, Linked Lists, Hash Tables, Search Trees

- Similar to stacks and queues there exist lock-free and lock-based implementations for many other concurrent DS
- The named are covered in the survey but not discussed here because of the lack of time

[1]

Priority Queues

- A concurrent priority queue is a data structure linearizable to a sequential priority queue that provides insert and delete-min operations with the usual priority queue semantics
- Lock-based implementation using fine grained locking organized like a heap
- Lock-free implementation using a concurrent skiplist

[1]

References

[1] M. Moir and N. Shavit, Concurrent data structures. In Handbook of Data Structures and Applications, D. Metha and S. Sahni Editors, pages 47-14 - 47-30, 2007. Chapman and Hall/CRC Press.

[2] Sun Microsystems Laboratories, People, an online version is available at http://labs.oracle.com/people/ (10.11.2010)

[3] M. Herlihy and N. Shavit, The Art of Multiprocessor Programming. 2008. Elsevier Inc.

Discussion