Brief Announcement: Implementing Multi-Word Atomic Snapshots on Current Hardware

Chris Purcell
University of Cambridge Computer Laboratory
15 JJ Thomson Avenue
Cambridge, UK, CB3 0FD
Chris.Purcell@cl.cam.ac.uk

Tim Harris
University of Cambridge Computer Laboratory
15 JJ Thomson Avenue
Cambridge, UK, CB3 0FD
Tim.Harris@cl.cam.ac.uk

Categories & Subject Descriptors: D.1.3
[Programming Techniques]: Concurrent Programming
General Terms: algorithms, design, performance
Keywords: non-blocking, lock-free, atomic snapshots

We present an algorithm that takes an atomic snapshot of a section of memory, without support from other code, using only read accesses to memory. Our goal is to ensure simultaneous read operations do not conflict at any point in the memory hierarchy, without inflating memory usage. Mutual-exclusion, even "multiple-reader single-writer" variants, and reference counting demand exclusive access to shared *reader counts* of some form. Systems using only local memory require frequent use of expensive read-after-write memory barriers, constraining their optimisation both in the CPU and the memory hierarchy. Other algorithms rely on an external garbage collector, delaying memory reuse and increasing their footprint in memory.

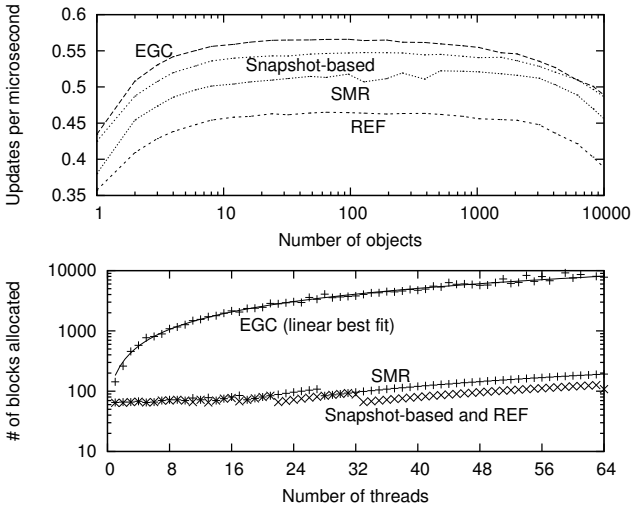
Our algorithm relies on the following observation: if a series of read instructions complete without needing to load memory into the cache, and without an intervening local write operation, then the reads can be viewed as occurring atomically at the start of the series. It is thus trivial to see that such a series of read instructions forms a multi-word snapshot. The snapshot algorithm simply loops until all reads hit in the cache. This is verified using values from the processor's internal cache-miss counter. To prevent an intervening local write by a pre-empting thread, we also count user/supervisor mode switches, which must occur during pre-emption.

```
do
  do
    // Start snapshot
    (m, s) := (#Cache-Misses, #Mode-Switches)

    Reads for snapshot
    Optional load-linked

  until (m, s) == (#C-Misses, #M-Switches)
until ( optional store-conditional succeeds )
```

Furthermore, the snapshot algorithm is lock-free *provided the cache can store all the data at the same time.* While not unreasonable, this may be a problem in situations where several memory locations in a pointer chain map to the same set in the cache, causing overflow even though there is enough capacity. This issue also applies to other proposals to support multi-word atomic operations in cache hardware.

On hardware that provides it, a load-linked can form part of the snapshot. We used this to implement an existing linked list [1] with a non-conflicting read operation. Each read is done in a single snapshot. We compared this algorithm to one relying on a large-footprint epoch garbage-collector (EGC), one using reference counting (REF), an adaptation by Maged Michael [2] (SMR), and a hybrid system using snapshots for reads and reference counting for memory management. Both of our snapshot-based algorithms performed within 5% of the fastest, EGC, and without its vastly inflated memory footprint.

We ran the tests on a dual-processor PowerPC machine (Apple's 2x1.25GHz G4). Currently, the only processor lines providing sufficient access to L2-miss counters are PowerPC and Itanium architectures.



REFERENCES

[1] Harris, T. A Pragmatic Implementation of Non-Blocking Linked Lists. In *Proceedings of the 15th International Conference on Distributed Computing.*

[2] Michael, M. High Performance Dynamic Lock-Free Hash Tables and List-Based Sets. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures.*

Copyright is held by the author/owner.
PODC'04, July 25–28, 2004, St. Johns, Newfoundland, Canada.
ACM 1-58113-802-4/04/0007.