Research trends in Multicore Programming

Contents of Lecture 12

- Power Efficient HTM
- Thread-level speculation
- C11Tester
- Relaxed lock-free data structures

Energy Efficient HTM

- By Do and Dubois from USC in Los Angeles 2016
- They found that over 42 % of aborted transactions have multiple aborts
- By delaying the restarting of aborted transaction, energy can be saved
- To each core a table is added which helps predict when it is better to delay a restart
- They found that up to 37 % energy could be saved this way
- `https://dl.acm.org/doi/10.1145/2875425`

HTM for speculative parallelization of for-loops

- By Salamanca, Amaral och Araujo, 2016 IEEE Trans par. distr. comp.
- Thread-level speculation (TLS) implemented with HTM
- Some loops cannot be analyzed by parallelizing compilers
- In software thread-level speculation normal hardware is used and data-races are detected and then the parallel execution of a loop is aborted
- This is normally difficult to make efficient

Evaluation on SPEC CPU2006

- SPEC CPU2006 contains single-threaded C/C++ and Fortran benchmarks
- It is an industry standard to evaluate compilers and CPUs
- First version from 1989 and current from 2017
- Some benchmarks were modified manually to run loops speculatively in parallel
- Four Intel and POWER8 CPUs were used and speedup of up to 3.8 was noted
- `https://ieeexplore.ieee.org/document/8038067`

A bug detector for C/C++ atomics

- There are no data-races between accesses to atomic variables
- How can bugs in lock-free data structures easily be found?
- Writing a few test cases is clearly insufficient
- Use stress tests which randomizes input and runs for hours.
- Is that sufficient?

A more systematic approach

- Evaluate all executions!
- Obviously not practical for millions of randomized inputs.
- Only a small number of test cases is realistic
- But how can all executions be evaluated???

C11Tester from UC Irvine

- The C11Tester is a library linked into the executable program
- Clang/LLVM instruments the application program with calls to C11Tester
- C11Tester has own functions for threads, mutex and atomics
- At runtime, C11Tester creates a graph showing the modification order of atomic variables
- C11Tester schedules the threads itself and not the Linux kernel

Using the modification order graph

- The graph can be used to see which executions have not yet been tested
- The graph could quickly become extremely huge
- The key idea is to remove parts of the graph using semantics of atomic operations
- Using the graph, C11Tester can guarantee that all executions are actually tested
- Hopefully there is one execution that actually triggers a failure
- Compare this to running the program millions of times until it crashes
- C11Tester can give much more useful information

`http://plrg.ics.uci.edu/c11tester`

Relaxed lock-free data structures

- Some algorithms require strictly working data structures.
- An RPN calculator which ”usually” pops the top of the stack is bad.
- Consider a queue and threads taking out work items and which prefer to take them in a FIFO order. It may be OK to take them out in a slightly different order.
- A shop with multiple cashiers and ”central” queue numbers (nummerlapp) typically is more fair than a supermarket with multiple queues, but both work OK.
- For threads taking out items from a queue, it may be OK if they get items close to the head of the queue.
- k-relaxed queue: remove takes one of the $k + 1$ first items

Semantic relaxtion of a queue

- Instead of always removing the oldest item, one of the $k + 1$ oldest items is removed.
- The idea is to let the queue consist of multiple internal queues (called subqueues) to avoid contention on a single queue.
- This research direction started in 2013 with a paper by Henzinger, Kirsch, Payer, Sezgin, and Sokolova: *Quantitative relaxation of concurrent data structures*
- Philippas Tsigas at Chalmers leads a world class research group on lock-free data structures including on this topic

Philippas Tsigas: 2D framework queue

- Tsigas' group created a 2D framework (2019) that can be used with different data structures including stack and queue.
- A removed node from a stack or queue has a *rank error* which is how far from the top/head the removed node was
- The idea is to specify at compile-time the number of subqueues / stacks. This is called the *width*.
- To bound the rank error, the number of queue puts and gets in different subqueues is kept within an interval, or window.
- This is to prevent one queue being too far away from another.
- How far away from each other they can be is called the *depth*.
- The width and depth together give a bound on the rank error.
- The width and depth are constants which the application developer can specify.
- For the queue there is one put and another get window.

More details

- The number of operations done so far, the width, and the depth define an interval or window.
- Each subqueue knows the current window.
- To do an operation, a thread tries to use the previous subqueue.
- If that is valid with respect to the window, it performs the operation on the same subqueue.
- This improves cache locality.
- If the previous was invalid, then a new queue is searched for. Initially a few random jumps and if none was found a linear search.
- If no valid queue was found it tries to move the window (by depth) and tries again.
- The max rank error of a 2D queue is (width - 1)$\times$depth.

Kåre von Geijer: Elastic relaxation

- A more flexible solution would be to let the application adjust the width and depth during runtime.
- This problem was solved in an efficient way by Kåre von Geijer in his MSc thesis from 2022 (Kåre studied engineering math at LTH).
- When the width or depth is changed, it only affects new insertions (no node redistribution which would take too much time).
- Changing the depth is done by a sequentially consistent atomic read-modify-write.
- To change the width, the put window is first adjusted and later the get window.
- When the put width is increased, a special gap node is inserted in a new subqueue, and when a get sees it, the get width is also increased.
- When the put width is decreased, a subqueue is frozen to not accept new nodes.

`https://lup.lub.lu.se/student-papers/search/publication/9092950`