Adding lock elision to Linux

Linux Plumbers Conference
Aug 2012
Andi Kleen
ak@linux.intel.com

TM programming models

- Transactional Memory programming is a new programming model
  - This talk is not about TM programming models
- This presentation is about accelerating existing programs with locks, not about writing software for a new model

Intel® Transactional Synchronization Extensions (Intel® TSX)

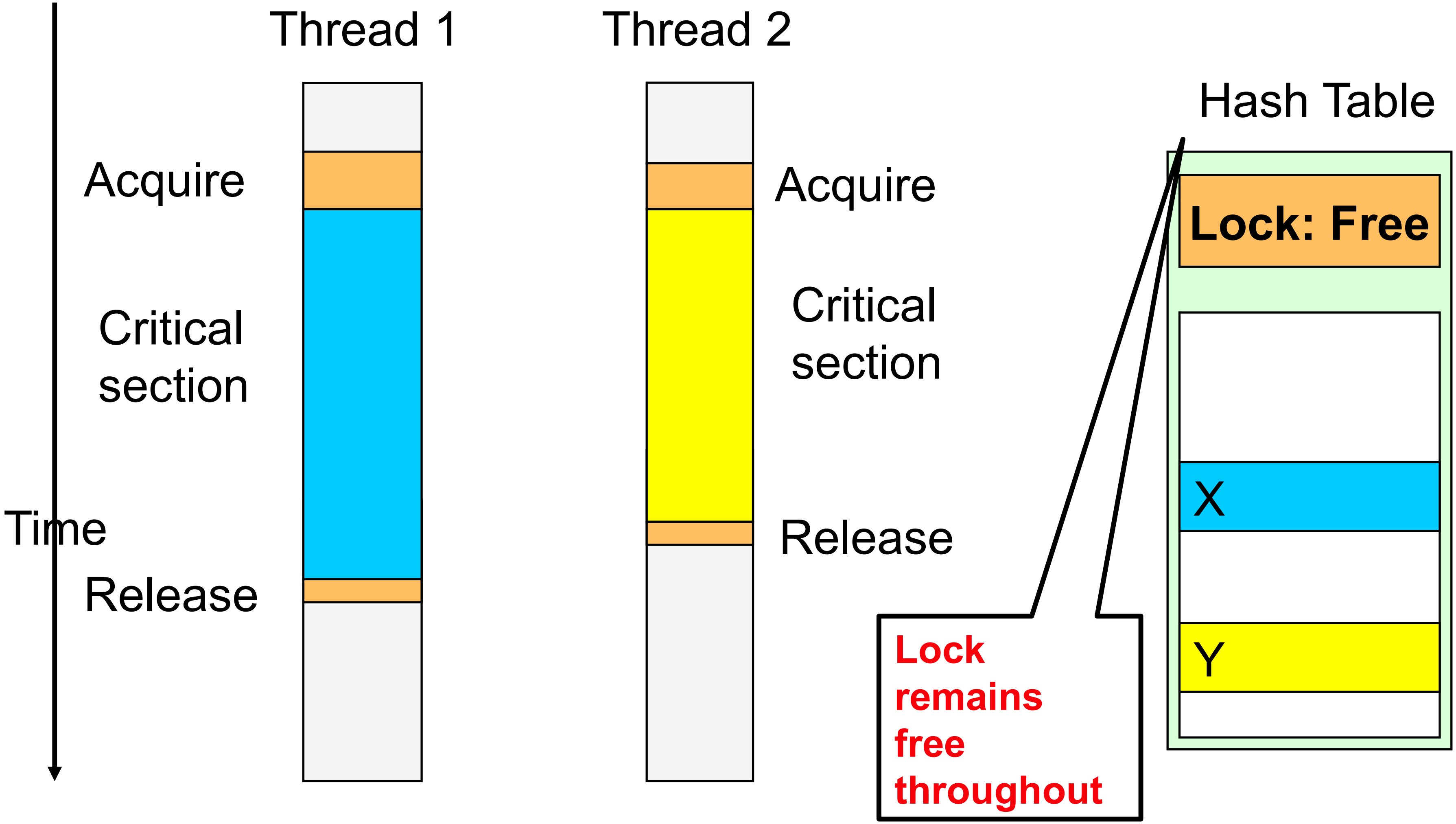
- Transactionally execute programmer-specified critical sections
  - If successful, perform atomic commit
  - If unsuccessful, rollback state/discard updates
- Focus on locking granularity optimizations
  - Goal: Fine-grain performance at coarse-grain effort

Goes beyond LOCK latency improvements to *expose parallelism* through lock elision

Interfaces to identify critical sections

- HLE uses XACQUIRE and XRELEASE prefixes
  - Legacy compatible hints, ignored on non TSX systems
  - Don’t acquire lock, execute sections speculatively
  - Hardware buffers loads and stores, checkpoints registers
  - Hardware attempts to commit atomically without locks
- RTM uses the XBEGIN and XEND instructions
  - Flexible interface
  - Similar operation to HLE, except:
  - Aborts transfer control to target specified by the XBEGIN operand
  - Abort information returned in a register
- XTEST & XABORT

Canonical elided execution



No serialization/communication if no data conflicts

Basic RTM elided lock

```
elided_lock(lock) {
    if (_xbegin() == _TXN_STARTED) {
        if (lock is free)
            /* puts lock into read set */
            /* execute lock region */
            return;
        _xabort(0xff);
        /* 0xff signals lock busy */
    }
    /* come here on abort */
    original locking code
}
```

```
elided_unlock(lock) {
    if (lock is free)
        _xend();   /* commit */
    else
        original unlocking code
}
```

- Simple wrapping code pattern
- Original lock code

Basic lock elision enabling

- Change existing lock library for elision
- Application is unchanged
  - With dynamic linking, no recompile needed

- Tune application for better elision success
  - Typically small changes
  - Optional, for better performance

POSIX Pthread mutex interface

```
pthread_mutex_lock(&mutex);
…. critical section….
pthread_mutex_unlock(&mutex);
```

Eliding in glibc pthread mutexes

- Glibc version that elides pthread mutexes
- Binary compatible. Any binary, that uses libc pthread locks today, can elide

- Currently supports pthread mutexes and pthread rwlocks
  - Only basic types: timed, not adaptive or recursive or robust
  - Elision can be controlled with environment variable (PTHREAD_MUTEX=…)
- Optional per lock annotation support in source

Successfully elided locks are:

- Scalable
- Non blocking
- Fine grained
- Not contended
- Without lock cache line bouncing
  - Can often dominate with small critical section
- Reader/Writer locks for free

What if elision aborts

- Can happen due to unsupported instructions, context switches, data conflict, overflow
  - See specification for full list
- When elision fails, lock will fall back to take the lock normally
  - In fact, everyone speculating on that lock will fall back
- Then, all the lock scaling problems appear
  - But you have a fast path that works around it
- But even abort may have non-intuitive benefit

Tuning programs

- Generally avoid costly aborts
- In general, standard “cache line locality” tuning to avoid conflicts
- Typically improves scaling without elision, too

Common abort problems

- False sharing
  - Add padding, as needed
- Global statistic counters inside locks
  - Remove or make per thread
- Re-writing unchanged shared data
  - Add check for data the same
- Syscalls/IO
  - Move out of lock or don't elide lock
- x87 usage on 32bit
  - Switch to SSE2

malloc

- Older glibc dlmalloc has high number of conflicts
- Can be fixed with "—enable-experimental-malloc" when building glibc
  - Default in glibc 2.15
  - Alternatively tcmalloc et.al. are elision friendly
- Other allocations may have similar problems

lock_is_locked()

- Lock appears free inside RTM region
  - Unlike HLE, where it appears locked
- Use _xabort() in lock_is_locked() to preserve semantics
  - Correct answer in non-speculation
- Some programs use it widely in assert
- Guarding assert with _xtest() avoids abort
  - Simple pattern that can be handled with semantic patches
  - assert(is_locked(l)) -> assert_is_locked(l)
  - assert_lock_is_locked(l) -> !_xtest() && assert(i_l(l))

More is_locked semantics

- pthreads does not have is_locked()

- But is_locked() can be emulated with try_lock()
  - Lock(l) if (!try_lock(l)) do_something
  - This changes semantics even in glibc pthreads
  - Not observed in the wild so far

Linux kernel is very scalable, but…

- Futex hash locks
- VM: LRU, zone locks, mm_sem, page table, anon vma chains
- Reclaims: i_mmap_mutex, tree_lock
- Slab locking
- Socket locks
- File system: i_mutex, journal locks
- Btrfs: extent cache, tree root lock
- RCU: write side locks
- Wait queue locks
- File locking
- Signal locks
- …
- The hot lock in your favorite workload

Kernel elision implementation

- Basic concept: elide kernel locks to improve kernel scalability
  - Benefits all applications
- Elide spinlocks/rwlocks/mutexes/rwsems/bit lock with elision wrapping pattern
  - Semaphores would work too, but rarely used now
- “Opt-out” strategy currently
  - Enable all locks with elision
  - Opt out a few strategic ones that do poorly

Paravirt ops for kernel elision

- Disabling/enabling interrupts abort
  - Common inside locks in the kernel
- Can use paravirt_ops patch mechanism
  - Supports patching interrupt enabling/disabling

- Add _xtest() to cli/sti to avoid aborts
- Also using it for adding the elided ticket locks
- For other locks, using binary patching to enable/disable

Kernel is_locked changes

- Not needed for correctness, BUG_ON aborts
  - Most can be done using semantic patches
  - No changes for calls outside lock
- BUG_ON(!spin_is_locked()) -> lockdep_assert_held()
- BUG_ON(!mutex_is_locked()) -> mutex_assert_held()
- … similar for other lock types
- Various uses are bugs (patches submitted)
- RCU debugging tests lock state and needs a few _xtest()s
  - In general, lock debugging does not elide though

References

- Specification http://software.intel.com/file/41604
- http://github.com/andikleen/glibc
  rtm-2.17 branch
  Work in progress glibc extension for RTM elision
- Kernel patches coming soon
- Feedback to ak@linux.intel.com

Backup

Terminology

- Transaction
  - Speculative state in CPU. Buffers memory operations
- Read-set / Write-set
  - All cache lines read/written in a transaction
- Conflict
  - Read-write conflict with other CPU (leads to abort)
- Abort
  - Transaction rolls back side effects
- Commit
  - Transaction state becomes atomically visible

Annotating pthread locks

- Should be used only rarely
  - And a lot to type...

PTHREAD_MUTEX_HLE_ADAPTIVE_NP	Force elision
PTHREAD_MUTEX_TIMED_NONHLE_NP	Force no elision

- pthread_mutex_t lock = PTHREAD_TIMED_NONHLE_MUTEX_INITIALIZER_NP;
- For allocated mutexes:

```
pthread_mutexattr_t attr;
pthread_mutexattr_init(&attr);
pthread_mutexattr_settype(&attr,  PTHREAD_..._NONHLE_NP);
pthread_mutex_init(&mutex, &attr);
```

Legal Information

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.
- Intel may make changes to specifications, product descriptions, and plans at any time, without notice.
- All dates provided are subject to change without notice.
- Intel is a trademark of Intel Corporation in the U.S. and other countries.
- *Other names and brands may be claimed as the property of others.
-