Portable atomic operations and lock-free synchronization

Hans-J. Boehm
HP Laboratories

© 2004 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice

Programming with Pthreads

- Programs are required to be “fully synchronized”.
  - Data writes may not happen concurrently with other accesses.
  - Typically pthread_mutex_lock() is used to ensure this.
    - Or perhaps barriers, or …
  - And this is good practice 98% of the time.
    - Programmers occasionally get it right.

- This talk is about the other 2% of the time

  …though it allows you to build more widely useful things.

The problems with locks and barriers

- Performance
  - For some algorithms.
- Signals
  - Locks and signals don’t play together.

Fully synchronized programs can be slow

- Traditional pthread_mutex's require:
  - Dynamic library call
  - 2 x (Atomic op + memory barrier)
- Sample cycle costs:

	cas	Mem barrier	lock/unl.
2.0 Xeon	124	125*	336
1.0 Itan.	10	4*	109
500 PIII	25	19*	156

*Not needed with compare-and-swap (cas)

Faster alternatives:

- Examples:
  - Reference counts (atomic add)
    - C++ standard libraries not fully synchronized
  - "Or" into "bit" vector
    - atomic byte store
    - Atomic-or into memory location
  - Hash table or cache read without locking
  - Extensive literature on lock-free algorithms

Faster alternatives: some measurements

- Sieve of Eratosthenes
  - Compute primes between 10K and 100M
  - Each thread executes:

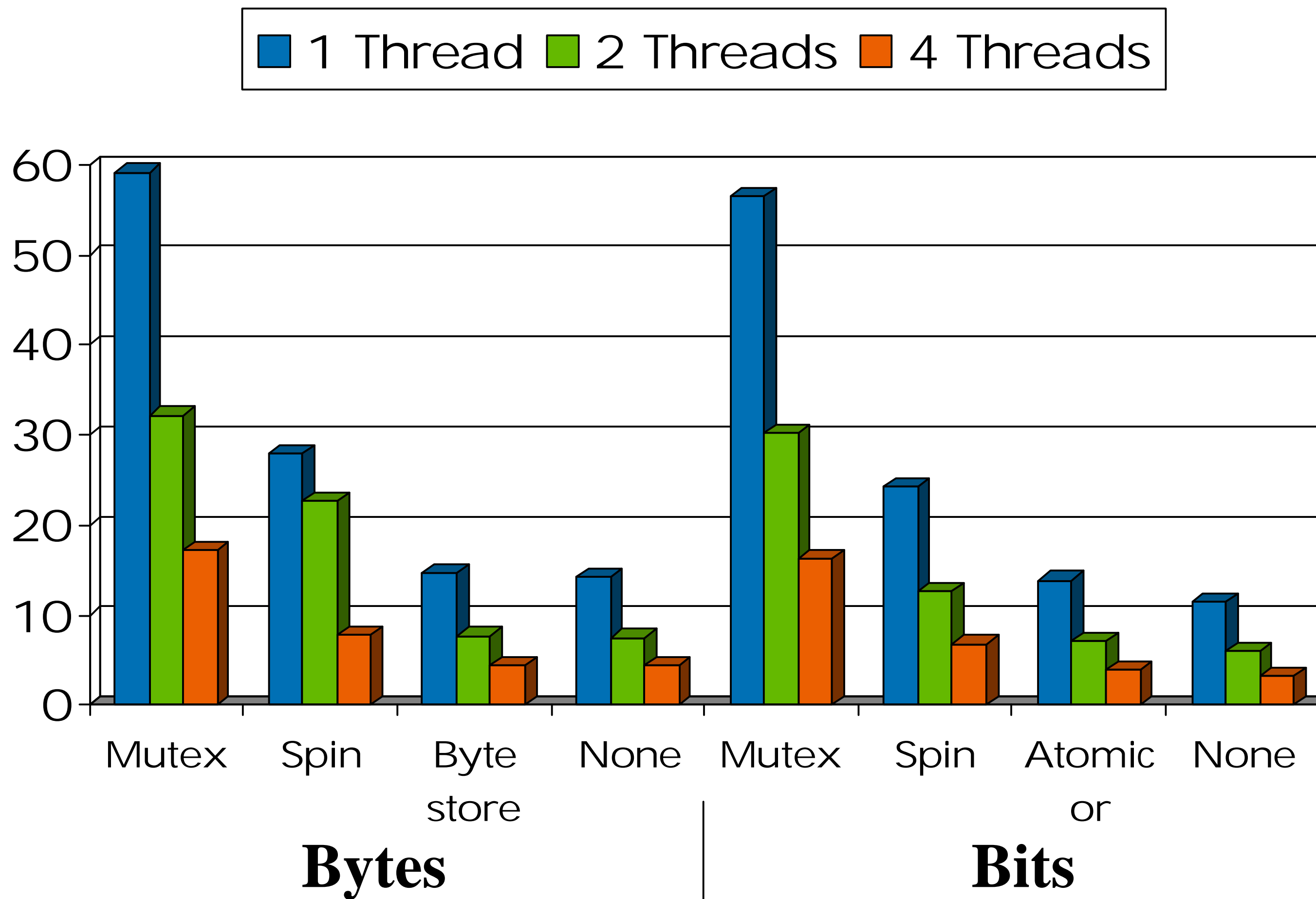
```
for (my_prime = start; my_prime < 10000; ++my_prime)
    if (!get(my_prime)) {
        for (multiple = my_prime; multiple < 100000000;
                multiple += my_prime)
            if (!get(multiple)) set(multiple);
    }
```

  - Get/set operate on 100M “bit” array.
  - Models part of our garbage collector.

Measurements: Alternatives

- Bit array vs. byte array
- Synchronization alternatives:
  - None (thread unsafe, but usually “works”)
  - Atomic byte stores for bytes
  - Atomic “or” into bit array
    - Needs portable access to e.g. compare-and-swap
  - Pthread_mutex locks (every 256 bits)
  - Pthread_spin locks (every 256 bits)

Running Time (secs, 4x1GHz Itanium 2)



Signals

- Pthread locks are not async-signal-safe.
  - Inherent problem: Interrupted thread may hold lock.
  - Signal handlers would like to update data structures.
    - E.g. a sampling profiler.
  - Lock-free data structures allow this, but need:
    - Memory consistency model.
    - Portable access to hardware synchronization.

Requirements for a solution

- A better standard …
  - … or a license to cheat
- Need to allow access to shared data outside locks.
- Access to hardware-provided atomic operations.

Atomic operations access

- Hardware provided atomic operations (fetch-and-add, compare-and-swap) vary.
- Associated barrier semantics vary.
- Many attempts to provide a portable layer.
- Most don’t pay enough attention to barrier semantics.

Atomic_ops: Our solution

- Provides operations consisting of pairs:
  - Atomic memory operation (nop, load, fetch_and_add …)
  - Ordering semantics (no barrier, full barrier, release, …)
- Large number of exported operations.
- Consistently named.
  - Many fewer things to remember.
- Most automatically synthesized from small description.
- Client specifies minimum requirements;
- Library uses cheapest sufficient implementation.

Example: Correct lazy initialization

```
if (!AO_load_acquire_read(&is_initialized)) {
   // Lock and recheck if unsafe to reinitialize.
   object_to_initialize = initial_value;
   AO_store_release_write(&is_initialized, 1);
}
// safe to read object_to_initialize here.
```

- Generates barriers on Alpha, ld.acq, st.rel on Itanium, compile-only barrier on X86.

Current work:

- Library of interesting, portable lock-free data structures.
- Example:
  - Efficient linked stack of preallocated objects.
  - Allows simple memory allocation from signal handler.
  - Lock-free with double-word compare-and-swap
    - (e.g. future Itanium, X86-32, Intel's X86-64)
  - "Close enough" with single-word compare-and-swap.
  - For details, talk to me, or wait for PODC paper.

Applications:

- Used in qprof user level profiler: http://www.hpl.hp.com/research/linux/qprof/
  - Allows safe update of shared profiling data from signal handler
  - Atomic_ops implementation available from above page.
- Used by libunwind.
- Will be used in our garbage collector.
- Hopefully other low-level performance-critical or signal/interrupt-based libraries…

Backup slides

- The rest isn't really part of this talk.

Pthreads memory model

"Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it. Such access is restricted using functions that synchronize thread execution and also synchronize memory with respect to other threads…"