Lecture 8 — C++ Atomics, Compiler Hints, Restrict

Patrick Lam & Jeff Zarnett
p.lam@ece.uwaterloo.ca, jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 10, 2017

Locks and Atomics

Atomics are a lower-overhead alternative to locks as long as you're doing suitable operations.

Remember that what we wanted sometimes with locks and mutexes and all that is that operations are indivisible.

Ex: an update to a variable doesn't get interfered with by another update.

Remember the key idea is: an atomic operation is indivisible.
Other threads see state before or after the operation; nothing in between.

About C++ atomics

You can use the default `std::memory_order`.
(= sequential consistency)

Don't use relaxed atomics unless you're an expert!

Different Memory Options

- memory_order_acquire
- memory_order_release
- memory_order_acq_rel
- memory_order_consume
- memory_order_relaxed
- memory_order_seq_cst

Really, don’t use C++ relaxed atomics!

C++ atomics: Key Idea

An *atomic operation* is indivisible.

Other threads see state before or after the operation, nothing in between.

Simplest: `atomic_flag`

```
#include <atomic>

atomic_flag f = ATOMIC_FLAG_INIT;
```

Represents a boolean flag.

Operations on `atomic_flag`

Can clear, and can test-and-set:

```
#include <atomic>

atomic_flag f = ATOMIC_FLAG_INIT;
int foo() {
  f.clear();
  if (f.test_and_set()) {
    // was true
  }
}
```

`test_and_set`: atomically sets to true,
returns previous value.

No assignment (=) operator.

Although I guess in C++ you could define one if you wanted.

This is kind of a dangerous thing about C++.

If in C you see a line of code like `z = x + y;` you can have a pretty good idea about what it does and you can infer that there's some sort of natural meaning to the + operator there, like addition or concatenation.

In C++, however, this same line of code tells you nothing unless you know...
(1) the type of x,
(2) the type of y, and
(3) how the + operator is defined on those two operands *in that order*.

But I'm digressing.

Using more general C++ atomics

Declaring them:

```
#include <atomic>

atomic<int> x;
```

Libary's implementation:

- on small types, lock-free operations;
- on large types, mutexes.

What to do with Atomics

Kinds of operations:

- reads
- writes
- read-modify-write (RMW)

Reads and Writes

C++ has syntax to make these all transparent:

```
#include <atomic>
#include <iostream>

std::atomic<int> ai;
int i;

int main() {
    ai = 4;
    i = ai;
    ai = i;
    std::cout << i;
}
```

Can also use `i = ai.load()` and `ai.store(i)`.

Read-Modify-Write (RMW)

Consider `ai++`.

This is really

```
tmp = ai.read(); tmp++; ai.write(tmp);
```

Hardware can do that atomically.

Other RMWs: `+-`, `&=`, `etc`, `compare-and-swap`

more info:
`http://preshing.com/20130618/atomic-vs-non-atomic-operations/`