A P R I L 1 3 , 2 0 1 0

ATOMIC PROPOSAL, N1452

BLAINE GARST

APPLE INC.

blaine@apple.com

Introduction

There is a simple language formulation of much that is available directly from hardware with regard to synchronization of operations across multiple threads of control. The industry has spent considerable effort providing various forms of grouped control primitives for making coordinated changes to multiple memory locations, by way of mutexes and condition variables.

An interesting opportunity lies untapped, however, in the form of single memory object operations that is quite useful.

The vast majority of modern multi-cpu architectures provide atomic compare-and-swap, test-and-set, or other forms of atomic operations on memory. These individual instructions coordinate with the caches to do correct behavior to the backing memory. Across processors, however, memory read and memory store barriers must be used to see these results reliably.

The proposal is to add an __atomic type qualifier to volatile marked memory and to have it have the following effects:

1. all +=, /=, ^= etc. assignment operators are overloaded to provide correct atomic behavior, either directly via hardware instructions or possibly by small compare-and-swap code sequences as the implementation decides, so that accumulation uses can be directly expressed in the language. Thus, even float identifiers can be supported as accumulators. Accumulation is an interesting form of distributed computing that allows computation to be distributed widely and provides a natural gathering of the arithmetic results.

2. __atomic identifiers require memory read barriers (as necessary on the hardware) for every read so that stale prefetched results can be discarded, and memory-store barriers such that correct communication with other threads is assured given proper scheduling of thread activities.

3. Initialization of an atomic identifier requires definition since visibility of this identifier to other threads is vital to correct distributed behavior. This can be accomplished by requiring an initializing expression or by decreeing an initial value per type for otherwise uninitialized objects. Requiring initializing expressions is preferred.

4. Implementations are free to not implement this facility by causing an error at translation time for any operation not supported. This provides a guarantee that if it compiles it works as best the hardware can provide.

5. A proposed ?= : tertiary assignment operator is introduced to express the compare-and-swap hardware primitive. A library routine could do as much.

Given the implicit memory barriers of atomic variables many of the proposed library additions to the standard become moot, yet should still be considered since this proposal is not yet in wide use at the time of this writing, yet may be by the time the standard goes to vote. Consider this an improved industry practice.

Modifications

The following are the more formal descriptions of changes necessary to the **N1425** proposed draft standard to reflect the preceding high level descriptions of atomic.

Section 5.1.2.4

para5: The __atomic type qualifier designates objects as being atomic and as requiring acquire, release, and acquire-modify-release operations directly on these objects. Additonally, the library....

1. Section 6.2.5

Para 26. Any type so far mentioned...

[[[this should move to become para 20 I think]]] there is no const/volatile/restrict function (or closure) type.

A further qualifier implying volatile is __atomic, forming an atomic object.

2. Section 6.2.6.1

When a value is stored into an atomic object it must be done such that the store is done via release semantics, e.g. a memory-store-barrier is required. When an atomic object is read it must be done via an acquire operation, e.g. a memory-read-barrier must be issues as necessary on the hardware architecture.

3. Section 6.4.1,

Add __atomic to the list of keywords.

4. Section 6.4.6

Add ?= to list of punctuators.

5. Section 6.5.2.4

Postfix ++ and -- operators on atomic objects must guarantee that the object is correctly modified w.r.t. all other threads, and that the resulting value was that which was modified by the expression.

6. Section 6.5.3.1

Prefix ++ and -- operators on atomic objects must guarantee that the object is correctly modified w.r.t. all other threads, and that the resulting value was that which resulted from the expression.

7. Section 6.5.16.2 Compound Assignment

Compound assignment of an atomic object requires that the value modified be that which is visible to all threads as the value of the object, and that all threads shall see this result for all other atomic operations including reads and further assignment modifications. If such assignment cannot be accomplished a translation error must be issued.

Note that

```
__atomic int x = 1; // global

x *= 2;  // some thread

x = x*2;  // not the same as x *= 2 if other threads manipulate x
```

8. Section 6.5.16.3 The tertiary assignement operator ?= :

A new tertiary operator ?= : tests that the lvalue has current value offered in the first expression and if so assigns the value provided by the second expression, yielding a boolean value affirming or denying that the assignment occurred. A memory-store-barrier is issued such that a successful conditional store may be used as a guard in a mutex style.

[Thus could be expressed as a new library function or function-style operator instead of via tertiary syntax.]