M A Y 1 0 , 2 0 1 0
Draft 5

ATOMIC PROPOSAL, N1473

BLAINE GARST

APPLE INC.

blaine@apple.com

Introduction

The C language community has for decades explored concurrency in the form of threads, mutexes and condition variables to describe all grain sizes of parallelism, and these are now reflected in the proposed C0x Standard. Also reflected is a new formalism for describing the memory model and in particular a new atomics library for describing operations on designated atomic memory objects.

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.

What is missing is direct language treatment of sequentially-consistent atomic behavior for C language objects.

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

1. all compound assignment operators (e.g. +=, /=, ^= , ...) are overloaded to provide sequentiallly-consistent atomic behavior, whether achieved directly via hardware instructions , by small inline compare-and-swap code sequences, or by support from compiler intrinsics, as the implementation decides, so that accumulator 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. Yet even so pure accumulation is today rare compared to the use of small integers as indices into larger data structures, and so rather than simply aquire-release we require full sequential-consistency.

2. Ordinary read and write access to __atomic identifiers is sequentially-consistent, requiring memory-load barriers as necessary on the hardware before every read, and memory-store barriers after writes so that dependent data can be properly provided.

3. A new ?= : tertiary assignment operator is introduced to express an assign value if identifier has the specific value operation. When used upon an atomic qualified identifier, a sequentially-consistent compare-and-swap operation will be performed. This primitive can be used, for example, to implement a simple spin-lock that guards changes to other objects.

__atomic applies uniformly to scalars, arrays, and aggregates, yet has little to no meaning for function results and parameters. An atomic structure would be read and written atomically using intrinsic helper functions, members not marked __atomic use unchanged access rules.

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 sequentially-consistent operations directly on these objects. Additionally, 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 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 sequentially-consistent semantics, e.g. a memory-store-barrier is required. When an atomic object is read it must also be done via a sequentially-consistent operation, e.g. a memory-load-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 sequential-consistency for that object 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 sequential-consistency for that object 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 sequential-consistency. The value modified must 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 qualified reads and further atomic operations.

Note that

```
__atomic volatile 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 assignment operator ?= :

A new tertiary operator *lvalue ?= candidateExpr : replacementExpr* tests that the *lvalue* has *candidateExpr* value and if so assigns the *replacementExpr* value, yielding a boolean value affirming or denying that the assignment occurred. If lvalue is marked atomic, the test and assignment must be performed with sequentially-consistent behavior, e.g. the test-and-set must be done atomically, with memory-load and memory-store barriers.

9. Section 6.7.3 Type Qualifiers

Paragraph 1: Add __atomic to the list.

10 7.16.3 para 2, 3;

spelling of "heed" is wrong.