n2279 - Proposal to make aliasing consistent

Rationale:

The limitation of lvalue access by types was not properly completed or clarified in the Standard. Neither temporary change of effective type for “type punning” or type assignment/erasure which is necessary for allocators were clearly specified. The exception for character pointers was added when it was realized that the Standard would otherwise completely prohibit implementation of memcpy. It still appears to be impossible to write an efficient "memcpy" without optimizer intervention or to write a memory allocator at all in conforming C with the existing Standard. For example: consider an allocator of device local memory in a GPU and how freed objects of one type could be combined or divided and reassigned a new effective type when allocated again. The current Standard appears to consider malloc/free as primitives, not as C implementable library functions. Furthermore, common operations in longstanding common practice, such as using an int type to compute a checksum on a message structure may be forbidden.

Part of the lack of clarity comes from an apparent contradiction in the Standard. Section 6.3.2.3.7 permits a pointer to an object type to be converted to a pointer to a different object type and 6.3.2.1 permits any pointer to be converted to a void pointer type and then converted to any other type but the current language in 6.5.7 seems to imply that dereferencing those pointers may be undefined behavior even if the pointers are properly aligned - which would make the conversion useless at best.

This change is not a complete cure. A memory allocator needs to be able to consolidate or divide objects and the Standard provides no mechanism for that process other than customary practice. Access to the underlying binary representation of a type is a fundamental capability of C programming.

Some of the motivation for the restrictions on aliasing is in order to make vectorization more practical, but there are less complex routes towards helping compilers identify opportunities for such optimizations. In fact, the current approach appears to have made “restrict” a neglected sideline.

Proposed change

6.5 Expressions

7 An object shall have its stored value accessed only by an lvalue expression that has one of the following types:88)

— a type compatible with the effective type of the object,

— a qualified version of a type compatible with the effective type of the object,

— a type that is the signed or unsigned type corresponding to the effective type of the object, — a type that is the signed or unsigned type corresponding to a qualified version of the effective type of the object,

— an aggregate or union type that includes one of the aforementioned types among its members (including, recursively, a member of a subaggregate or contained union), or

—–a character type, or

— a properly aligned pointer of any object type that has been explicitly converted from a valid pointer of the effective object type as specified in 6.3.2.3.7.