isbits Union Optimizations

In Julia, the `Array` type holds both "bits" values as well as heap-allocated "boxed" values. The distinction is whether the value itself is stored inline (in the direct allocated memory of the array), or if the memory of the array is simply a collection of pointers to objects allocated elsewhere. In terms of performance, accessing values inline is clearly an advantage over having to follow a pointer to the actual value. The definition of "isbits" generally means any Julia type with a fixed, determinate size, meaning no "pointer" fields, see `?isbitstype`.

Julia also supports Union types, quite literally the union of a set of types. Custom Union type definitions can be extremely handy for applications wishing to "cut across" the nominal type system (i.e. explicit subtype relationships) and define methods or functionality on these, otherwise unrelated, set of types. A compiler challenge, however, is in determining how to treat these Union types. The naive approach (and indeed, what Julia itself did pre-0.7), is to simply make a "box" and then a pointer in the box to the actual value, similar to the previously mentioned "boxed" values. This is unfortunate, however, because of the number of small, primitive "bits" types (think `UInt8`, `Int32`, `Float64`, etc.) that would easily fit themselves inline in this "box" without needing any indirection for value access. There are two main ways Julia can take advantage of this optimization as of 0.7: isbits Union fields in types, and isbits Union Arrays.

isbits Union Structs

Julia now includes an optimization wherein "isbits Union" fields in types (`mutable struct`, `struct`, etc.) will be stored inline. This is accomplished by determining the "inline size" of the Union type (e.g. `Union{UInt8, Int16}` will have a size of two bytes, which represents the size needed of the largest Union type `Int16`), and in addition, allocating an extra "type tag byte" (`UInt8`), whose value signals the type of the actual value stored inline of the "Union bytes". The type tag byte value is the index of the actual value's type in the Union type's order of types. For example, a type tag value of `0x02` for a field with type `Union{Nothing, UInt8, Int16}` would indicate that an `Int16` value is stored in the 16 bits of the field in the structure's memory; a `0x01` value would indicate that a `UInt8` value was stored in the first 8 bits of the 16 bits of the field's memory. Lastly, a value of `0x00` signals that the `nothing` value will be returned for this field, even though, as a singleton type with a single type instance, it technically has a size of 0. The type tag byte for a type's Union field is stored directly after the field's computed Union memory.

isbits Union Arrays

Julia can now also store "isbits Union" values inline in an Array, as opposed to requiring an indirection box. The optimization is accomplished by storing an extra "type tag array" of bytes, one byte per array element, alongside the bytes of the actual array data. This type tag array serves the same function as the type field case: its value signals the type of the actual stored Union value in the array. In terms of layout, a Julia Array can include extra "buffer" space before and after its actual data values, which are tracked in the `a->offset` and `a->maxsize` fields of the `jl_array_t*` type. The "type tag array" is treated exactly as another `jl_array_t*`, but which shares the same `a->offset`, `a->maxsize`, and `a->len` fields. So the formula to access an isbits Union Array's type tag bytes is `a->data + (a->maxsize - a->offset) * a->elsize + a->offset`; i.e. the Array's `a->data` pointer is already shifted by `a->offset`, so correcting for that, we follow the data all the way to the max of what it can hold `a->maxsize`, then adjust by `a->ofset` more bytes to account for any present "front buffering" the array might be doing. This layout in particular allows for very efficient resizing operations as the type tag data only ever has to move when the actual array's data has to move.

« SubArrays System Image Building »