New, mo’ bettah attributes Ticket #304

Squyres and Tillier

Madrid, September 2013

Last edit: v0.8, 11 Sep 2013

Current attribute system sucks

- Complex rules for inter-language interop
  - Including deprecated, truncating Fortran routines
- Only used on some handle types
- Big ambiguities in attribute callback definitions
  - See Microsoft June 2013 Forum presentation
  - Fixing the current system is… difficult
- Best solution is a new system

Goals of new system

- Cache on any handle type
- Key objects have a proper handle type
- Key objects are global (vs. per handle type)
- Simple language interop between Fortran/C
- Clearly defined callback semantics
- Deprecate old keyval/attribute functions

Design Overview / Goals

- "Key = value" system
  - Keys are ref-counted handles: MPI_Key
  - Values are MPI_Aint *(no translation between C/ Fortran)*
- Values can be cached on *any* MPI handle type
- Callbacks when handles are:
  - Copied
  - Freed
  - Destroyed (this is new)

MPI-3 p103:30 Advice to Users: Do we care?

- C users may be tempted to avoid the usage of MPI_GET_ADDRESS and rely on the availability of the address operator &. Note, however, that &cast-expression is a pointer, not an address. ISO C does not require that the value of a pointer (or the pointer cast to int) be the absolute address of the object pointed at -- although this is commonly the case. Furthermore, referencing may not have a unique definition on machines with a segmented address space. The use of MPI_GET_ADDRESS to “reference” C variables guarantees portability to such machines as well.

Probably not. This is more about pointer math than lossless translation pointer ←→ MPI_Aint. Bill will check the standard to be sure, but is pretty sure.

Overview

- Deprecate all old attribute functions
- New pre-defined key handles:
  - MPI_KEY_NULL
  - MPI_KEY_TAG_UB
  - MPI_KEY_HOST
  - MPI_KEY_IO
  - MPI_KEY_WTIME_IS_GLOBAL

Key functions

- MPI_Key_create
- MPI_Key_free
- MPI_Key_c2f
- MPI_Key_f2c

- …just like many other MPI handle types

Key functions: create

- MPI_Key_create(value_copy_fn, value_free_fn, value_destroy_fn, MPI_Aint context, MPI_Key *key)
  - Function pointers for when the handle is copied, freed, and destroyed (prototypes discussed later)
  - Context to be passed to callbacks
- Once created, this key can be used with *any* handle type
  - MPI_Comm, MPI_Datatype, MPI_Key, ...etc.

Key functions: free and c2f/f2c

- As you would expect
  - MPI_Key_free(MPI_Key *key)
  - MPI_Key MPI_Key_f2c(MPI_Fint fkey)
  - MPI_Fint MPI_Key_c2f(MPI_Key ckey)

Value functions

- MPI_Value_set
- MPI_Value_get
- MPI_Value_clear
  - For explicitly unsetting a value on a handle

Value functions: set

- MPI_Value_set(MPI_Key key, int handle_type, MPI_Handle handle, MPI_Aint value)
  - handle_type: indicates type of handle (e.g., MPI_HANDLE_COMM, MPI_HANDLE_DATATYPE, …)
  - handle: type that allows any handle
    - MPI_COMM_WORLD, MPI_INT, my_comm, etc.
  - value: MPI_Aint of the value
- If a value was already set on (key, handle)
  - Behaves as if MPI_VALUE_CLEAR was first called to clear the current value before new value is set

Value functions: get

- MPI_Value_get(MPI_Key key, int handle_type, MPI_Handle handle, MPI_Aint *value, int *flag)
  - key, handle_type, handle: Same as VALUE_SET
  - value: pointer to MPI_Aint
  - flag: logical, indicates whether a value is set on this handle / was filled into the “value” OUT param

Value functions: clear

- MPI_Value_clear(key, handle_type, handle)
  - key, handle_type, handle: Same as VALUE_SET
  - Explicitly clear a value on a handle such that MPI_VALUE_GET on that (key, handle) will return flag==false
  - Not an error to clear a (key, handle) for which no value is set

Key callbacks

- All return void
- Do not affect the semantics of invoking MPI functions
  - E.g., MPI_COMM_FREE functionality is not impacted by the return of a value callback
  - Because otherwise, you get deferred destruction craziness (see prior Microsoft presentation)

Key callbacks: Copy

- Function params
  - key
  - handle_type: enum indicating type of handle
  - old_handle
  - new_handle
  - key_context: from Key_create
  - old_value: value set on old handle
  - new_value: value to be set on new handle
  - flag (OUT): logical, indicates whether a new value was set

Key callbacks: Free

- Invoked at MPI_*_FREE time
- Function params
  - key
  - handle_type: enum indicating type of handle
  - handle
  - key_context: from Key_create
  - value: value currently set on handle

Key callbacks: Free

- Invoked for every value set on the handle
  - Before the handle is freed
  - It's ok to use the handle in the callback

```
MPI_Comm_free(comm)
 → invokes my_callback()

my_callback(…, comm, …) {
   MPI_Send(…, comm);
}
```

✓ This is ok!

Key callbacks: Free

- Calling MPI_*_FREE on the handle in the callback is erroneous

```
MPI_Comm_free(comm)
 → invokes my_callback()

my_callback(…, comm, …) {
   MPI_Comm_free(comm);
}
```

❌ This is NOT ok!

Key callbacks: Free

- Can call MPI_*_FREE on *another* handle in a free callback
  - Well-defined, but can lead to reentrancy issues due to recursion
  - Advice to Users: this is complicated; don't do it

```
MPI_Comm_free(comm)
 → invokes my_callback()

my_callback(..., comm, ...) {
    MPI_Comm_free(othercomm);
}
```

✓ This is ok… but risky

Key callbacks: Free

- If Key was previously MPI_KEY_FREEd
  - key argument will be MPI_KEY_NULL
  - But the key context will still be valid

Key callbacks: Free

- Can VALUE_GET on this handle in this callback
  - VALUE_GET(this_key, this_handle) → flag == false
  - VALUE_GET(other_key, this_handle) → flag == ?
    - Ordering of free callbacks is non-deterministic
- Cannot VALUE_SET or VALUE_CLEAR any key *on this handle* in this callback

  → <u>This is under debate.</u> Other option is to allow VALUE_SET/VALUE_CLEAR in callback and queue up future free callbacks

Key callbacks: Destroy

- Invoked:
  - When corresponding MPI object is destroyed
  - Before MPI_VALUE_CLEAR returns
  - Before MPI_VALUE_SET replaces an old value
- Value destroy function params
  - key
  - handle_type: enum indicating type of handle
  - handle
  - key_context: from Key_create
  - value: value currently set on handle

Key callbacks: Destroy

- If Key was previously MPI_KEY_FREEd
  - key argument will be MPI_KEY_NULL
  - But the key context will still be valid

- (just like free callback behavior)

Performance implication

- By allowing attributes on any MPI handle type:
  - Can cache values on an MPI_Request
  - MPI_TEST/MPI_WAIT may invoke attribute callbacks
  - This (probably) adds a single extra "if" statement in the TEST/WAIT completion code path
- All other handle types invoke callbacks at MPI_*_FREE time
  - Performance implications largely irrelevant

KTHXBYE