Argonne NATIONAL LABORATORY

Function Classification

Wesley Bland
Argonne National Laboratory

Aurélien Bouteiller
University of Tennessee

June 6, 2013

ICL UT

U.S. DEPARTMENT OF ENERGY

Motivation

- Reading/Interpreting the standard can be difficult for everyone (writers, implementers, users)
- Determining whether functions are collective, local, blocking, etc. is not always as easy as it seems.
- Some function definitions make this very clear:

MPI_WIN_FENCE(assert, win)

IN	assert	program assertion (integer)
IN	win	window object (handle)

```
int MPI_Win_fence(int assert, MPI_Win win)

MPI_Win_fence(assert, win, ierror) BIND(C)
    INTEGER, INTENT(IN) ::  assert
    TYPE(MPI_Win), INTENT(IN) ::  win
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_WIN_FENCE(ASSERT, WIN, IERROR)
    INTEGER ASSERT, WIN, IERROR
```

The MPI call MPI_WIN_FENCE(assert, win) synchronizes RMA calls on win. The call is collective on the group of win. All RMA operations on win originating at a given process and started before the fence call will complete at that process before the fence call returns. They will be completed at their target before the fence call returns at the target. RMA

- Others do not:

MPI_COMM_FREE(comm)

INOUT	comm	communicator to be destroyed (handle)

```
int MPI_Comm_free(MPI_Comm *comm)

MPI_Comm_free(comm, ierror) BIND(C)
    TYPE(MPI_Comm), INTENT(INOUT) ::  comm
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_COMM_FREE(COMM, IERROR)
    INTEGER COMM, IERROR
```

This collective operation marks the communication object for deallocation. The handle is set to MPI_COMM_NULL. Any pending operations that use this communicator will complete normally; the object is actually deallocated only if there are no other active references to it. MPI_COMM_FREE will succeed if passed a revoked communicator (as defined in Section 14.3.1). This call applies to intra- and inter-communicators. The delete callback functions for all cached attributes (see Section 6.7) are called in arbitrary order.

Advice to implementors. A reference-count mechanism may be used: the reference count is incremented by each call to MPI_COMM_DUP or MPI_COMM_IDUP, and decremented by each call to MPI_COMM_FREE. The object is ultimately deallocated when the count reaches zero.

Though collective, it is anticipated that this operation will normally be implemented to be local, though a debugging version of an MPI library might choose to synchronize. (*End of advice to implementors.*)

???

Implications

- In addition to confusing users…
- Defining the behavior of a category of functions is not straightforward
  - Easy to miss defining behavior for some functions without specifically enumerating each function
- Example:
  - Defining failure semantics for blocking vs. non-blocking functions
    - Which functions are actually blocking?
    - Oops…I forgot that RMA is non-blocking

Proposal

- Add to the list of semantic terms in Section 2.4
  - Matched
  - Unmatched
  - Global
- Define new semantics to be added to each function definition
  - Specify which terms in 2.4 apply to the function
- Simpler than rewriting the definition of each function to be more clear, but accomplishes the same goal.

Classifications

- Old
  - Nonblocking, Blocking, Local, Non-local, Collective
- New
  - Matched
    - A procedure is matched if it requires participation at multiple ranks, including the calling rank. It is not required that the matching calls be identical (such as in the case of MPI_SEND and MPI_RECV).
  - Unmatched
    - A procedure is unmatched if it is initiated by one rank, but has an effect on at least one other rank without a participating call by the other rank(s).
  - Global
    - A procedure is global if it is called by all ranks, regardless of communicator. A global operation may not be synchronizing.

Function Classification

- Each function gets at least 3 classifications
  - Local/non-local
  - Blocking/non-blocking
  - Matching/non-matching
- Additional classifications if appropriate
  - Collective
  - Global

Example Function Definition

Old Version

MPI_COMM_DUP(comm, newcomm)

IN	comm	communicator (handle)
OUT	newcomm	copy of comm (handle)

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)

MPI_Comm_dup(comm, newcomm, ierror) BIND(C)
    TYPE(MPI_Comm), INTENT(IN) ::  comm
    TYPE(MPI_Comm), INTENT(OUT) ::  newcomm
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_COMM_DUP(COMM, NEWCOMM, IERROR)
    INTEGER COMM, NEWCOMM, IERROR
```

New Version

MPI_COMM_DUP(comm, newcomm)

IN	comm	communicator (handle)
OUT	newcomm	copy of comm (handle)

CLASSIFICATION: non-local, blocking, matching, collective

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)

MPI_Comm_dup(comm, newcomm, ierror) BIND(C)
    TYPE(MPI_Comm), INTENT(IN) ::  comm
    TYPE(MPI_Comm), INTENT(OUT) ::  newcomm
    INTEGER, OPTIONAL, INTENT(OUT) ::  ierror

MPI_COMM_DUP(COMM, NEWCOMM, IERROR)
    INTEGER COMM, NEWCOMM, IERROR
```