Argonne NATIONAL LABORATORY

One-sided Communication in MPI-3

Pavan Balaji

Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY

What's needed in MPI-3 RMA?

- MPI-2.2: Elegant interface, but cumbersome to use in many cases
- Some things can be fixed with changes in semantics
- Some things might require additional calls
- Goals:
  - Should be able to implement application algorithms used with other one-sided models in a natural manner (no jumping through hoops)
  - ARMCI/GA should work natively on top of MPI-3 RMA
  - UPC, Chapel should be able to use MPI-3 RMA as the runtime system
  - Application models such as work-stealing and master-worker data management should be easy to implement

Proposed Changes

- **Proposal 1: Remote Completion Semantics (and associated RMA calls)**
- Proposal 2: Additional RMA functionality
- Proposal 3: Minor (subtle) updates to semantics (backward compatible)

Remote Completion Semantics

- Currently opening/closing an epoch is tied to completion of requests initiated within that epoch
  - We need to close an epoch to use the result of an RMA operation
- Idea is to decouple epochs from remote completion
  - Very similar to two-sided communication (operations have requests)
    - User can wait/test for the completion of one or more requests
    - MPI_Wait, Waitall, etc., are used for local completion
    - MPI_Win_swait, swaitall, etc., are used for remote completion
  - Model:
    - Iput + compute + wait + compute + swait
    - Put + compute + swait
    - Sput

Communication Calls

- Deprecate MPI_PUT, MPI_GET, MPI_ACCUMULATE for:
  - PUT: MPI_WIN_IPUT, MPI_WIN_PUT, MPI_WIN_SPUT
  - GET: MPI_WIN_IGET, MPI_WIN_GET
  - ACC: MPI_WIN_IACC, MPI_WIN_ACC, MPI_WIN_SACC
- IPUT, PUT, IGET, IACC, and ACC use a request argument that can be used to wait for local and/or remote completion
- Current MPI implementations (e.g., MPICH2) try to "guess" whether PUT should be initiated immediately or wait
  - Explicit blocking calls remove this guess work

Remote Completion Calls

- MPI_WIN_SWAIT: Waits on a request
- MPI_WIN_SWAITALL: Waits on an array of requests
- MPI_WIN_SWAITSOME: Waits on an array of requests
- MPI_WIN_SWAITANY: Waits on an array of requests
- MPI_WIN_SWAIT_TARGET: All RMA operations to a target have completed
- MPI_WIN_SWAIT_ALLTARGET: All RMA operations to all targets have completed
- (Equivalent STESTs for all the above functions)

Proposed Changes

- Proposal 1: Remote Completion Semantics (and associated RMA calls)
- **Proposal 2: Additional RMA functionality**
- Proposal 3: Minor (subtle) updates to semantics (backward compatible)

MPI_WIN_COPY and MPI_WIN_RMW

- MPI_WIN_COPY (and ICOPY, SCOPY)
  - Third-party RMA
  - Useful if MPI-3 is used as the runtime system for UPC
    - Communication with shared pointers
  - Master-worker models where the master can manage data for the workers
- MPI_WIN_RMW (and IRMW)
  - Read-modify-write functionality
  - Fetches the original copy of the data to the origin

User-defined operations in Accumulate/RMW

- Operations allowed in ACC/RMW are restricted
  - Proposal: Allow for user-defined operations like Reduce
- MPI_WIN_REGISTER_OP
  - Collective call for everyone to register their local operation
  - Accumulate/RMW with an OP execute the locally registered OP on the target
- Useful to reset the target buffer, for example
  - MPI_WIN_PUT cannot do this without transferring the entire data
- If we miss a feature, users do not need wait till MPI-4 for it to be fixed

Extensions to MPI_WIN_LOCK/UNLOCK

- Three types of locks:
  - MPI_WIN_SHARED_LOCK
    - Concurrent lock user semantics
    - Permits non-conflicting RMA operations. Load/store not allowed.
  - MPI_WIN_DIRECT_LOCK
    - (TODO: query function to tell the user whether this is concurrent or not; not added as this might need to go into WIN_CREATE; discussion later)
    - Currently left as non-concurrent (basically exclusive like, but cache-coherent systems can internally optimize by adding concurrency).
    - Permits non-conflicting RMA operations. Load/store allowed.
  - MPI_WIN_EXCLUSIVE_LOCK
    - Non-concurrent lock user semantics
    - Permits conflicting RMA operations. Load/store allowed.

Group lock/unlock

- Additional functions for MPI_WIN_LOCKLIST and MPI_WIN_LOCKALL
  - Allows implementations to do a broadcast of LOCK requests, which is more efficient than the user calling individual LOCKs.
- Note that this functionality is conditional upon a change in the semantics discussed later!

Optional Locks

- MPI-2.2 uses locks both for concurrency and completion
- This proposal decouples these two and adds separate calls for completion, so the user can handle it explicitly
  - However, if the user does not explicitly wait for completion, then unlock will need to handle completion (MPI-2.2 requirement; can be relaxed in MPI-3).
- If the user handles completion separately, and does not need concurrency support from MPI, LOCK/UNLOCK basically do nothing: Should we make locks optional in such cases?
  - Pros: convenient for users
  - Cons: cases where it is safe to ignore locks might be hard to explain?
  - Cons: only convenience, very little performance benefit

MPI_Win_create (currently backward compatible)

- MPI_Win_create
  - Additional info arguments: “no_direct” and “no_excl”
  - “no_direct” allows all systems to not acquire explicit locks internally
  - “no_excl” allows cache-coherent systems to not acquire explicit locks internally
- What’s missing?
  - For the “no_direct” case, lock/unlock can be optional for the user. We cannot do this now since “info” is only a hint.
  - Need an argument which says whether I can have overlapping windows or not (can’t just say undefined for MPI_WIN_WORLD)
  - Need an argument which says whether direct locks are concurrent or not (cache-coherent systems can make it concurrent)

Proposed Changes

- Proposal 1: Remote Completion Semantics (and associated RMA calls)
- Proposal 2: Additional RMA functionality
- **Proposal 3: Minor (subtle) updates to semantics (backward compatible)**

Other minor semantic changes

- Concurrent RMA operations to the same location are defined as "erroneous"; they should be "undefined"
- MPI_REPLACE (operation in ACC) should be extended to work with RMW so as to return the original content of the buffer (equivalent to a SWAP)
- New MPI_COMP_AND_SWAP operation for ACC/RMW
- Distinct access epochs for a window must be disjoint "only" for active target (doesn't make sense for lock/unlock): needed to allow LOCKALL and LOCKLIST functionality
- MPI_WIN_LOCK should not only block for local windows, but also for remote windows if a process can do direct load/store on that window (e.g., in a shared memory buffer)
- LOCK/UNLOCK should not need MPI_ALLOC_MEM for portability; it's OK to need it for performance.

Usage Example: Shared locks (3 processes)

```
MPI_Win_create(..., MPI_COMM_WORLD, MPI_WIN_SHARED_LOCK, &win);
MPI_Win_lock_all(MPI_WIN_SHARED_LOCK, 0, win);
MPI_Barrier(MPI_COMM_WORLD);

if (rank == 0) {
    /* Put data X on window and ask rank 2 to read it */
    MPI_Win_sput(..., target_rank = 1, ..., win);
    MPI_Send(..., dest_rank = 2, ...);

    /* Read data Y put by rank 2 on window */
    MPI_Recv(..., dest_rank = 0, ...);
    MPI_Win_get(..., target_rank = 0, ..., win);
}
if (rank == 2) {
    /* Read data X put by rank 0 on window */
    MPI_Recv(..., source_rank = 0, ...);
    MPI_Win_get(..., target_rank = 1, ..., win);

    /* Put data Y on window and ask rank 0 to read it */
    MPI_Win_sput(..., target_rank = 0, ..., win);
    MPI_Send(..., dest_rank = 0, ...);
}

MPI_Win_unlock_all(win);
MPI_Barrier(MPI_COMM_WORLD);
MPI_Win_free(win);
```

Usage Example: Work Stealing (using local data)

```
MPI_Win_create(..., MPI_COMM_WORLD, MPI_WIN_DIRECT_LOCK, &win);

/* Check for work on window */
MPI_Win_lock(MPI_WIN_EXCLUSIVE_LOCK, target_rank = 1, 0, win);
MPI_Win_get(..., target_rank = 1, ..., win);

... read data got from window to see if there is work ...

... update data to take the work we want and put the rest back ...

MPI_Win_sput(..., target_rank = 1, ..., win);
MPI_Win_unlock(target_rank = 1, win);

MPI_Win_free(win);
```

Usage Example: Work Stealing (using only remote data)

```
MPI_Win_create(..., MPI_COMM_WORLD, MPI_WIN_DIRECT_LOCK, &win);
MPI_Win_register_op(op, &win); /* Op checks for work */

/* Check for work on window */
MPI_Win_lock(MPI_WIN_EXCLUSIVE_LOCK, target_rank = 1, 0, win);
MPI_Win_rmw(..., target_rank = 1, ..., op, win);
MPI_Win_unlock(target_rank = 1, win);

MPI_Win_free(win);
```